VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Webový informační systém BIGGIE

Web Information System BIGGIE

2010

Milan Trávníček

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava*.

Na základě dohody s firmou XEVOS Solutions s.r.o., pod kterou byl systém vyvíjen, nebudou vybrané části specifických postupů probírány do podrobna. Dále se zavazuji, že práci nebudu nikde zveřejňovat.

V Ostravě 23. dubna 2010 .....

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 23. dubna 2010 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli.

Abstrakt

Hlavním cílem diplomové práce je realizace skladového informačního systému jako webové aplikace. Práce má zachytit, navrhnout a implementovat základní firemní procesy obchodu a skladové evidence. Systém je postaven na technologii ASP.NET využívající platformu .NET Framework 3.5, implementován v programovacím jazyce C# a komunikuje s relačním databázovým systémem Microsoft SQL Server. Základním motivačním prvkem je využití knihoven AJAX k přiblížení webové aplikace stylu aplikace desktopové.

Klíčová slova: ASP.NET, .NET framework, AJAX, informační systém, webová aplikace

Abstract

The main goal of this diploma thesis is realization of warehouse information system as a web application. Purpose of work is to analyze, design and implement essential enterprise processes of business and data warehouse. System is build on the ASP.NET technology using .NET Framework 3.5 platform, implemented in C# programming language and communicates with the relational database system Microsoft SQL Server. The primary motivation is use AJAX libraries to approach web application UI to Windows-based application.

Keywords: ASP.NET, .NET framework, AJAX, information system, web aplication

Seznam použitých zkratek a symbolů

ADO	–	ActiveX Data Objects
AJAX	–	Asynchrynous JavaScript And Xml
ASP	–	Active Server Pages
BLL	–	Business Logic Layer
BO	–	Business Object
DAL	–	Data Access Layer
DOM	–	Document Object Model
ER	–	Entity Relationship
ERP	–	Enterprise Resource Planning
HTML	–	HyperText Markup Language
MVC	–	Model View Controller
ODS	–	ObjectDataSource
PDF	–	Portable Document Format
SQL	–	Structured Query Language
XHTML	–	eXtensible Hypertext Markup Language
XML	–	eXtensible Markup Language

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

S nástupem průlomové technologie ASP.NET AJAX došlo k myšlence navrhnout informační systém dostupný přes webový prohlížeč. V podstatě šlo o eliminaci základního chování webové aplikace, a to odesílání a zpětné načítání stránky při každém požadavku. Právě díky AJAX funkcionalitě nedochází k načítání celé stránky při obsluze požadavku, ale pouze vybrané části stránky bez efektu obnovení. Navíc se během komunikace pracuje s menším obsahem dat. Pokud tedy mluvím o přiblížení k desktopové aplikaci, mám na mysli využití této vlastnosti.

Práce byla vyvíjena pod záštitou firmy XEVOS Solutions s.r.o. sídlící ve Vědecko-technologickém parku Ostrava a stala se jádrem produktu XEVOS ERP Live. Jde o komplexní podnikový informační systém třídy ERP určený malým a středním podnikům, které vyžadují kontrolu nad firemními procesy. Zásadní výhodou je již zmíněná webová aplikace, díky které odpadají problémy s nasazením a uživatelé mohou přistupovat k firemním datům v reálném čase.

Podílel jsem se na vývoji jako součást tří členného týmu. Můj účel práce zahrnoval sestavení ER diagramu, návrh databáze, realizaci základních funkcí informačního systému skladu a také funkcí konkrétních určených zadavatelem během vývoje systému. Základní funkci představuje realizace agend pro faktury, dodací listy, objednávky, sklady a karty. Mezi konkrétní funkce patří např. převody mezi doklady, likvidování faktur, vytváření dceřiných karet a další.

Počátek dokumentu se věnuje přehledu klíčové technologie pro návrh systému. Obsah dalších kapitol slouží pro popis mnou realizovaných částí produktu XEVOS ERP Live. Jejich souhrn poskytl funkční rámec, z kterého vznikla verze webového informačního systému pod pracovním názvem BIGGIE. Dále bude prezentována pouze tato část produktu jako autonomní systém, neboť celkový popis produktu XEVOS ERP Live nespadá do účelu mé práce.

2 AJAX

Stavební technologií pro návrh systému byla ASP.NET postavená nad frameworkem .NET 3.5. Popisem těchto obecně známých technologií se dokument nebude věnovat. Dále tak popis jazyka C#, ve kterém byla aplikace implementována, nebude nastíněn.

Účel kapitoly spočívá v poskytnutí stručného přehledu ASP.NET AJAX Extensions knihovny, která staví na JavaScriptových knihovnách použitých v ASP.NET webových formulářích nebo v ASP.NET MVC aplikací. Součástí AJAX Extensions je také knihovna AJAX Control Toolkit obsahující desítky připravených ovládacích prvků.

AJAX zahrnuje technologie pro:

- účelnou prezentaci založenou na standardech XHTML a CSS (Cascade Style Sheet)
- dynamické zobrazování a interakce pomocí Document Object Model (DOM)
- hodně využívanou výměnu dat a jejich zpracování pomocí XML a XSLT
- asynchronní získávání dat pomocí XMLHttpRequest
- propojení předcházejících technologiích pomocí JavaScriptu

Využití AJAX

V místech, kde probíhá častá komunikace uživatele se serverem na jedné stránce, se hodí využít technologii AJAX. Použití AJAXu umožňuje provést základní validaci uživatelského vstupu již v prohlížeči (pomocí JavaScriptu), odeslat data na server, přijmout a popřípadě i zpracovat odpověď a nakonec vhodně změnit obsah stránky. Všechny tyto akce mohou probíhat na pozadí, proto jeho název vychází ze slova asynchronní [2].

2.1 Aplikační rámce pro AJAX

Programování v AJAXu představuje nutnost řízení objektu `XmlHttpRequest` dle instance prohlížeče a psaní opakujících se JavaScriptových kódů. Aplikační rámce (frameworky) zapouzdřují tyto kódy do použitelných částí, přičemž v dnešní době jsou velmi rozšířené a dostupné. AJAX frameworky lze rozdělit na serverové a klientské s určením specifického jazyku, pro nějž byly vytvořeny. Přehled vybraných rámců ukazuje tabulka 1:

Aplikační rámec	Popis	Klient/ Server	Specif. jazyk
DOJO	- nabízí funkci `bind()`, která nahrazuje řízení odesílání dat přes objekt XmlHttpRequest - podpora tlačítek `Vpřed` a `Zpět` - velmi stabilní	klient	JavaScript
TIBET	- nejstarší rámec, nastavba nad XmlHttpRequest - podpora webových služeb, podpora protokolů nižší úrovně - obsahuje nástroje pro práci s Google, Amazon - navíc nabízí vývojové prostředí pro prohlížeč, které umožňuje ladění a testování	klient	JavaScript
qooxdoo	- kombinace XMLHttpRequest, Iframe a skriptů - klientské funkce typu nabídky, okna s nápovědou, podpora techniky drag & drop	klient	JavaScript
jQuery	- obecně nejpoužívanější knihovny	klient	JavaScript
DWR (Direct Web Remoting)	- umožňuje volání metod Javy z webových stránek pomocí JavaScriptu - lze přirovnat k Struts či Tapestry - vyžaduje modifikaci konfiguračního souboru	server	Java
Rails (Ruby on Rails)	- excelentní vestavěná podpora AJAXu - obsahuje knihovny JavaScriptových funkcí a modul pro volání Ruby z JavaScriptu	server	Ruby
SWATO (Shift Web App. TO)	- pracuje v kontejneru od servletů 2.3 a výše - vyžaduje modifikaci konfiguračního souboru - výměna dat mezi serverem a klientem - JSON - volání metod Javy na serveru z prohlížeče - výchozí nastavení pro přístup služeb - Spring - textová pole s automatickým dokončováním	server	Java
ASP.NET AJAX	- umožňuje volání .NET metod z JavaScriptu na straně klienta, AJAX Extension - knihovna prvků AJAX Control Toolkit	server	C#, VB
SAJAX	- jedna knihovna pro volání metod nad potencionální jazyky - abstrakce většiny užívaných kódů	server	PHP,Perl ASP,Ruby Python
BACK-BASE	- Java Enterprise Ajax Framework - více než 250 miniaplikací a funkcí - integrace pro JSP, Struts, Spring, atd. - předávání dat mezi server-klient - JSON a XML	server	Java

Tabulka 1: Přehled aplikačních rámců pro AJAX

2.2 AJAX Extensions

V podstatě jde o sbírku serverových komponent, které využívají a mají v sobě vestavěnou knihovnu AJAX library - souhrn klientských skriptů (JavaSkriptů), jejichž hlavním účelem je zvýšení rychlost odezvy u webových aplikací. Tato knihovna je platformně nezávislá a integruje technologie AJAX a platformu .NET.

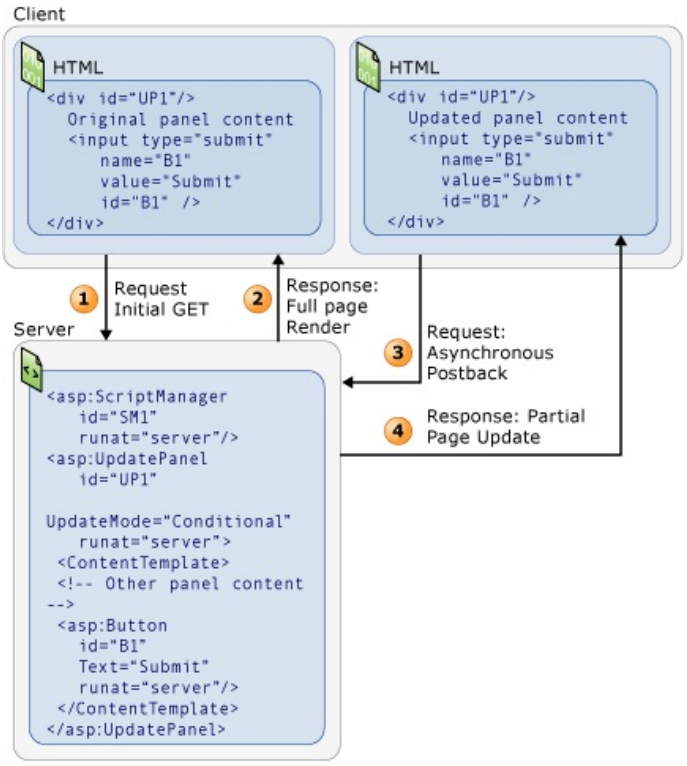
Základní prvky AJAX Extensions:

- **ScriptManager**
  Nevizuální ovládací prvek, který řídí všechny AJAXové prvky na stránce tím, že realizuje odkazy na JavaScriptové knihovny `AJAX Library`. Navíc může realizovat odkazy na další soubory vlastních skriptů. Zajistí tak částečné vykreslování a volání webových služeb. Musí být vložen pouze jeden na stránce s funkcionalitou AJAX a musí být umístěn před ovládacími prvky, jež má obsluhovat.
- **ScriptManagerProxy**
  Z důvodu jedinečnosti prvku `ScriptManager` na vzorové stránce (master page) a požadavků na jeho odlišnou konfiguraci v různých stránkách, došlo k návrhu prvku `ScriptManagerProxy`. Bývá vkládán do obsahových stránek, kde se s ním pracuje stejně jako s prvkem `ScriptManager`, se kterým bude poté spolupracovat a zajistí potřebné akce.
- **Timer**
  Vykonává asynchronní nebo synchronní zpětné odesílání stránky na server (postback) v definovaném intervalu. Využívá se nad prvkem `UpdatePanel`, kde provádí automatickou aktualizaci jeho obsahu.
- **UpdatePanel**
  Nevizuální ovládací prvek, jehož obsah v podobě ASP.NET nebo AJAX ovládacích prvků, bude odeslán při požadavku. Definuje tedy oblasti stránky pro částečné vykreslení bez postbacku.
- **UpdateProgress**
  Slouží pro zobrazení průběhu aktualizace ovládacího prvku `UpdatePanel`.

2.3 AJAX Control Toolkit

AJAX rozšiřuje ovládací prvky ASP.NET webového formuláře o automatické využití vlastností JavaScriptu realizovaných pomocí tzv. extenderů, které na ovládací prvek navážou prvky z Control Toolkit knihovny.

AJAX Control Toolkit obsahuje přes třicet ovládacích prvků připravených k použití na webové stránce. Například lze využít prvky pro zobrazení HTML editorů (HTML editors), textová pole s automatickým doplňováním, kaskádových seznamů (cascading dropdown lists) či modálních vyskakovacích oken (modal popup dialog boxes).



Obrázek 1: Proces částečného vykreslování stránky [4]

2.4 Využití ovládacího prvku UpdatePanel

AJAX Extensions zahrnuje nejpodstatnější ovládací prvek UpdatePanel, do jehož části `ContentTemplate` se vloží kód aspx stránky, který se bude částečně aktualizovat. Pro programové přidávání slouží vlastnost `ContentTemplateContainer`. UpdatePanel se aktualizuje v závislosti nastavení jeho vlastnosti `UpdateMode`.

Mód *Always* aktualizuje UpdatePanel při každém požadavku na celé stránce. Naopak mód *Conditional* aktualizuje UpdatePanel po splnění podmínky - požadavek vyvolal prvek UpdatePanelu, na něj vázaný spouštěč, explicitní volání metody `Update()` z kódu nebo se požadavek vyvolává z vnořeného UpdatePanelu. Zdali má být UpdatePanel aktualizován, v závislosti na požadavku z vnořeného UpdatePanelu, určuje jeho nastavení vlastnosti `ChildrenAsTriggers=true`.

Obrázek 1 ukazuje proces při prvním načtení stránky (kroky jedna a dva), kde dochází k jejímu celému vykreslení. Dále se, při zachycení události na tlačítku v UpdatePanelu, vyvolá asynchronní požadavek a v odpovědi se posílá a následně vykresluje u klienta pouze obsah v UpdatePanelu (kroky tři a čtyři). Manipulaci s vyhledáním značek v DOM struktuře stránky a jejich nahrazením aktualizovanými provádí klientská třída `PageRequestManager`.

UpdatePanel využívá UpdateProgress komponentu, které se na něj naváže přes vlastnost `AssociatedUpdatePanelID`. V případě delšího zpracovávání asynchronního požadavku dojde k zobrazení obsahu UpdateProgress prvku. Další vlastnost UpdatePanelu spočívá v jeho aktualizaci prvkem mimo jeho obsah. K tomu využívá oblasti `Triggers`, která registruje prvkem `AsyncPostBackTrigger` kontroler pro aktualizaci UpdatePanelu. V ukázce kódu 1 vídíme použití výše popsaných vlastností.

```
...

<asp:ScriptManager ID="ScriptManager1" runat="server" />
<asp:Button ID="Button1" Text="Aktualizuj UpdatePanel1" runat="server" />

...

<asp:UpdatePanel ID="UpdatePanel1" UpdateMode="Conditional" runat="server">
    <Triggers>
        <asp:AsyncPostBackTrigger ControlID="Button1" />
    </Triggers>
    <ContentTemplate>

        ...

        <asp:UpdateProgress ID="UpdateProgress1" AssociatedUpdatePanelID="UpdatePanel1"
             runat="server">
            <ProgressTemplate>
                 Aktualizuji  UpdatePanel1 ...
            </ProgressTemplate>
        </asp:UpdateProgress>

        ...

    </ContentTemplate>
</asp:UpdatePanel>

...
```

Výpis 1: Ukázka vázaní UpdateProgress a spouštěče na UpdatePanel

3 Specifikace požadavků

Webový informační systém BIGGIE poskytne přehled nad řízením a evidencí dat firmy zabývající se nákupem a prodejem, a s umístěním hlavního skladu v místě prodejny. Hlavní činnost systému se zakládá na významu informačního systému, tedy sběru, udržování, zpracování a poskytování informací a dat. Jedná se především o správu hlavního byznys procesu, tedy faktur, dodacích listů, objednávek, zákazníků, skladů, zboží a uživatelů do systému vstupujících. Zároveň se požaduje zachycení interakcí mezi prací uživatelů a vytvářenými daty.

Vzhledem k celkové robustnosti systému, zde nebudou podrobně probírány vedlejší funkční požadavky na systém. Vedlejší funkční požadavky si lze představit jako požadavky podporující provoz systému a doplňující jeho hlavní činnosti.

3.1 Soupis hlavních požadavků

Soupis jednotlivých funkčních požadavků je rozdělen podle záměru působnosti a obsahuje pouze informativní shrnutí požadovaných významných funkcí systému.

3.1.1 Evidence karet

Uživateli bude poskytnuto prostředí pro evidenci zboží, dále karet. Uživatel potřebuje, krom evidence základních údajů, přiřadit ke kartě informace typu - do jaké existující kategorie patří, na jakých skladech může být umístěna, či vybrání dodavatelů karty. Rozlišují se čtyři základní typy karet - obecná, mateřská, dceřiná a výrobek:

- **Karta mateřská**
  Karta mateřská vznikne z obecné karty, na kterou se odkáže karta typu dceřiná.
- **Karta dceřiná**
  Karta dceřiná vznikne odkazem na existující kartu s výběrem vlastností, které chce zdědit. Při změně hodnot v kartě mateřské dojde ke změně i v kartách dceřiných, které pod ní spadají.
- **Karta výrobek**
  Karta typu výrobek vznikne z obecné karty výběrem položek, které ji tvoří. Položkou může být existující karta nebo textový záznam s cenou.

3.1.2 Evidence výroby

Požadavek na výrobu obsahuje pouze základní funkce vyrobit a rozebrat, u kterých se tento proces bere jako nezávislý na čase a vztahuje se především k manipulaci s množstvím karet do výroby vstupujících. Správu výroby lze akceptovat pouze nad kartou typu výrobek.

Funkce vyrobení bude zahrnovat výběr zdrojového a cílového skladu, kde dojde u zdrojového skladu k ponížení množství karet výrobku a navýšení množství výrobku na cílovém skladě.

Funkce rozebrání bude zahrnovat nastavení cen jednotlivých karet výrobku, výběr zdrojového a cílového skladu, kde dojde u zdrojového skladu k ponížení množství karet výrobku a navýšení množství výrobku na cílovém skladě.

3.1.3 Evidence faktur

Správa faktur musí uchovávat a funkčně zachytit požadované typy faktur (hlavní, zálohová, dobropis) a jejich směr (přijatá/vydaná). Společným základem evidování faktury do systému jsou funkce pro přidání existující firmy na doklad, výběr karet a zobrazení seznamu dokladů s fakturou jakkoliv spojených. Pro správné vytvoření faktury je nutné poskytnout funkci pro nastavení zaokrouhlování dokladu, která se ovlivní další žádanou funkci zobrazení souhrnů faktury. Zobrazením souhrnů zahrnuje informace o celkové částce a přehled součtů základů jednotlivých dph. Uzavření faktury ukončuje celý obchodní proces.

Přijatá faktura obsahuje funkci pro automatické rozpočítávání nákladů. Uživatel vloží položku s hodnotou nákladu a ta se rozpočítá mezi nákupní ceny položek faktury podle vybraného způsobu přepočtů - podle kusů nebo podle poměru cena a množství. Další funkci, spadající do rozsahu faktury přijaté, lze nazvat oceňování karet, která se projeví při jejím platném uložení. Pro mazání či stornování faktury přijaté platí, že musí dojít ke zpětnému ovlivnění cen u karet.

Vydaná faktura poskytuje možnost zadání slev faktury. Slevy faktury mohou být zadány bez důvodu určení nebo podle druhu formy úhrady. Výsledná hodnota slevy se rozpočítá mezi ceny položek faktury a systém promítne použitou slevu do zobrazení souhrnů. Vydaná faktura dále umožňuje výběr jejího způsobu doručení a aplikuje řízení cenových skupin na prodejní ceny karet faktury.

- **Správa plateb**
  Správou plateb dostává uživatel prostor k uzavření faktury. Eviduje se forma úhrady platby (převodem/hotově) a také, zda byla platba vydána či přijata. K evidenci platby patří možnost přidání existující firmy na doklad, ale hlavní potřebou je svázání platby s vytvořenou fakturou, která čeká na uzavření, tzv. zlikvidováním. Takové svázání provede načtení informací o faktuře do evidované platby. Zlikvidováním neboli evidencí platby k celkové ceně faktury, dojde k uzavření faktury.

3.1.4 Evidence objednávek

U objednávky se musí evidovat pouze dva druhy dokladu určené směrem dokladu (přijatá/vydaná). Její hlavní funkce vycházejí z funkcí faktury. Obsahuje tedy funkce pro

přidání existující firmy na doklad, výběr karet, zobrazení seznamu dokladů s objednávkou jakkoliv spojených, nastavení zaokrouhlení, zobrazení souhrnů a u vydané objednávky také nabízí funkci určující způsob doručení. Vydaná objednávka umožňuje výběr jejího způsobu doručení.

3.1.5 Evidence dodacích listů

Základem dodacích listů je požadavek evidence naskladnění (příjemka) a vyskladnění (výdejka). Součástí vytvoření dodacího listu bude poskytnutí funkcí pro přidání existující firmy na doklad, výběr karet a zobrazení seznamu dokladů s dodacím listem jakkoliv spojených. Nutnou funkcí je výběr skladu, ke kterému se dodací list eviduje, a následná aktualizace množství karet na vybraném skladu. Ke každé kartě může uživatel zavést sériová čísla.

Pro evidování vráceného zboží na sklad bude poskytnuta funkce vytvoření vratky. Vratku lze vytvořit výběrem položek existujícího dodacího listu. Pro mazání a stornování dodacích listů platí, že musí dojít ke zpětnému ovlivnění množství karet na skladě.

3.1.6 Funkce převodů dokladů

Doklady typu faktura a dodací list lze mezi sebou převádět a vytvářet tak vazby mezi nimi. Jedná se o funkce vytvoření faktury z dodacího listu a naopak. Převod bude zajištěn v obou směrech dokladu, tedy jak přijatém, tak vydaném. Vytvoření převodu zahrnuje přenesení informací zdrojového dokladu na cílový s uchováním vazby mezi doklady. Vztah mezi doklady bude realizován dvěma způsoby - převodem s položkami a převodem bez položek. Při převodu s položkami se položky zdrojového dokladu převedou na doklad cílový. Vázané doklady budou zobrazeny u každého dokladu.

3.2 Ostatní požadavky na evidenci

Další požadované spravování záznamů ukazuje následující souhrn evidencí:

- **Evidence skladu**
  Smyslem správy skladů je umožnit uživateli evidovat dva typy skladů - fyzický a virtuální. Virtuální sklad obsahuje základní informace včetně vedoucího skladu a slouží pro kategorizování karet. Fyzický sklad navíc uchovává adresu, ovšem v rámci systému je jejich funkce společná, a to uchovávání počtů kusů karet v něm obsažených.
- **Evidence firem**
  Další potřebu vyvolává evidence zákazníků (firem), se kterými je firma užívající systém v pravidelném obchodním kontaktu. K uložení základních údajů o firmě

je nutné navíc evidovat, do jaké cenové skupiny firma patří. To ovlivní nastavení faktury vydané a objednávky přijaté při přidání firmy na doklad.

- **Evidence cenových skupin**
  Uživateli musí být nabídnuta evidence cenových skupin do systému. Cenová skupina určuje, k jaké nákupní ceně se vztahuje a o kolik bude cena navýšena. Sledují se tři typy nákupních cen - poslední, průměrná a průměrná vážená.

- **Evidence osob**
  Systém bude nabízet základní evidenci uživatelů určujících přístup do systému. Ukládání osob může uživatel využít k uchovávání kontaktů firmy, u toho vytvoření nebudou k osobě vyplňovány přihlašovací údaje.

3.3 Ostatní funkce

- **Společné funkce pro výpisy**
  Všechny výpisy budou poskytovat funkce pro vyhledávání záznamů a výběr zobrazených sloupců.

- **Výpis karet**
  Výpis karet bude navíc obsahovat vhodné zobrazení hierarchie v případě výskytu mateřské karty a bude podléhat aktuálnímu výběru kategorie, který bude také zobrazen.

- **Vytvoření PDF souborů**
  Vytvoření PDF souborů je funkcí obsaženou nad výpisy modulů faktur, dodacích listů a karet. Požadavek na vytvoření PDF souboru bude obsahovat přehledové zobrazení nad seznamem vybraných záznamů anebo detailní zobrazení nad jedním vybraným záznamem.

- **Úvodní statistické přehledy**
  Uživateli budou poskytnuty úvodní statistické přehledy jako např. souhrn množství karet, nejprodávanější karty, souhrny fakturací a jiné dle aktuálních požadavků.

- **Evidence změn záznamů**
  Nad výše jmenovanými správami bude, u jejich záznamů, evidována osoba, která záznam vytvořila, a která jej naposledy změnila, a to s časem evidence. Dále se požaduje pouze zneplatnění záznamů při požadavku funkce smazání, na rozdíl od jejich přímého vymazání.

- **Přihlášení a odhlášení**
  Základní funkčnost přihlášení a odhlášení bude doplněna o funkci automatického odhlašování při nečinnosti uživatele po dohodnutou dobu.

3.4 Role

Práce nad systémem bude umožněna maximálně deseti uživatelům vystupujících v rolích administrátor, vedoucí, nákupčí, prodejce, skladník, produktový a CRM manager.

- **Administrátor**
  Uživatel v roli administrátora může v podstatě provádět jakoukoliv funkci poskytovanou systémem. Úplné řízení se zavádí z důvodu nutnosti výchozího nastavení aplikace, provádění změn nad veškerými daty či celkové správy systému. Za výchozí nastavení aplikace se minimálně považuje zaevidování uživatele s rolí vedoucí, aby mohl získat přístup k práci nad systémem.

- **Prodejce**
  Uživatel pod rolí prodejce provádí vystavování faktur vydaných, objednávek přijatých a podílí se na správě cenových skupin a plateb. Zároveň má přístup pouze k prohlížení výdejek z důvodu jejich převodů na faktury vydané a provádí u karet úpravy částek pro cenové skupiny, které se projeví při prodeji.

- **Nákupčí**
  Uživatel pod rolí nákupčí provádí vystavování faktur přijatých, objednávek vydaných a podílí se na správě cenových skupin a plateb. Zároveň má přístup pouze k prohlížení příjemek z důvodu jejich převodů na faktury přijaté.

- **Skladník**
  Skladník se stará o vedení skladu, tedy eviduje příjemky a výdejky do systému. Další činností je vytváření těchto skladů, řízení výroby nebo vytváření karet s výběrem skladů, na kterých může být karta umístěna. Zároveň má omezený přístup k fakturám pro převod faktur na dodací listy.

- **Produktový a CRM manager**
  Produktový a CRM manager provádí evidování záznamů karet a firem do systému. Odpovídá především za správné zařazení karet mezi kategorie a dodavatele. Dále zodpovídá za správně zadané a aktuální informace k firmám v systému, které jsou poté využívány k přidání na doklady.

- **Vedoucí**
  Vedoucí má v podstatě také plnou kontrolu nad systémem, ovšem jeho účelem práce v systému je evidování pracovníků pro přístup do systému, podílí se na návrhu cenových skupin a evidování plateb do systému, sleduje pohyby cen karet a jejich množství na skladech. Dále se podílí na evidenci informací o firmách, zvláště zařazování firmy do cenové skupiny. Provádí také na kartě úpravy částek cenových skupin, které se projeví při prodeji. Pro statistické informace má k dispozici přehledy a analytiky. Modul analytik není zahrnut mezi požadavky vzhledem k robustnosti systému.

3.5 Nefunkční požadavky

Základním nefukčním požadavkem na systém je jeho návrh jako webová aplikace komunikující s databází SQL SERVER 2008. Vývoj bude prováděn nad technologií ASP.NET s využitím trial verzí komponent Infragistics a Telerik, kde si zadávající firma XEVOS Solutions s.r.o. vybere nejvhodnější komponentu, na kterou uplatní licenci.

3.6 Use-Case modely

Případy užití popisují funkce systému vzhledem k aktérům, kteří se daných funkcí účastní. Podkapitola zachycuje jak případ užití pro celkový popis systému, tak pro další podstatné funkce v systému.

3.6.1 Hlavní případ užití

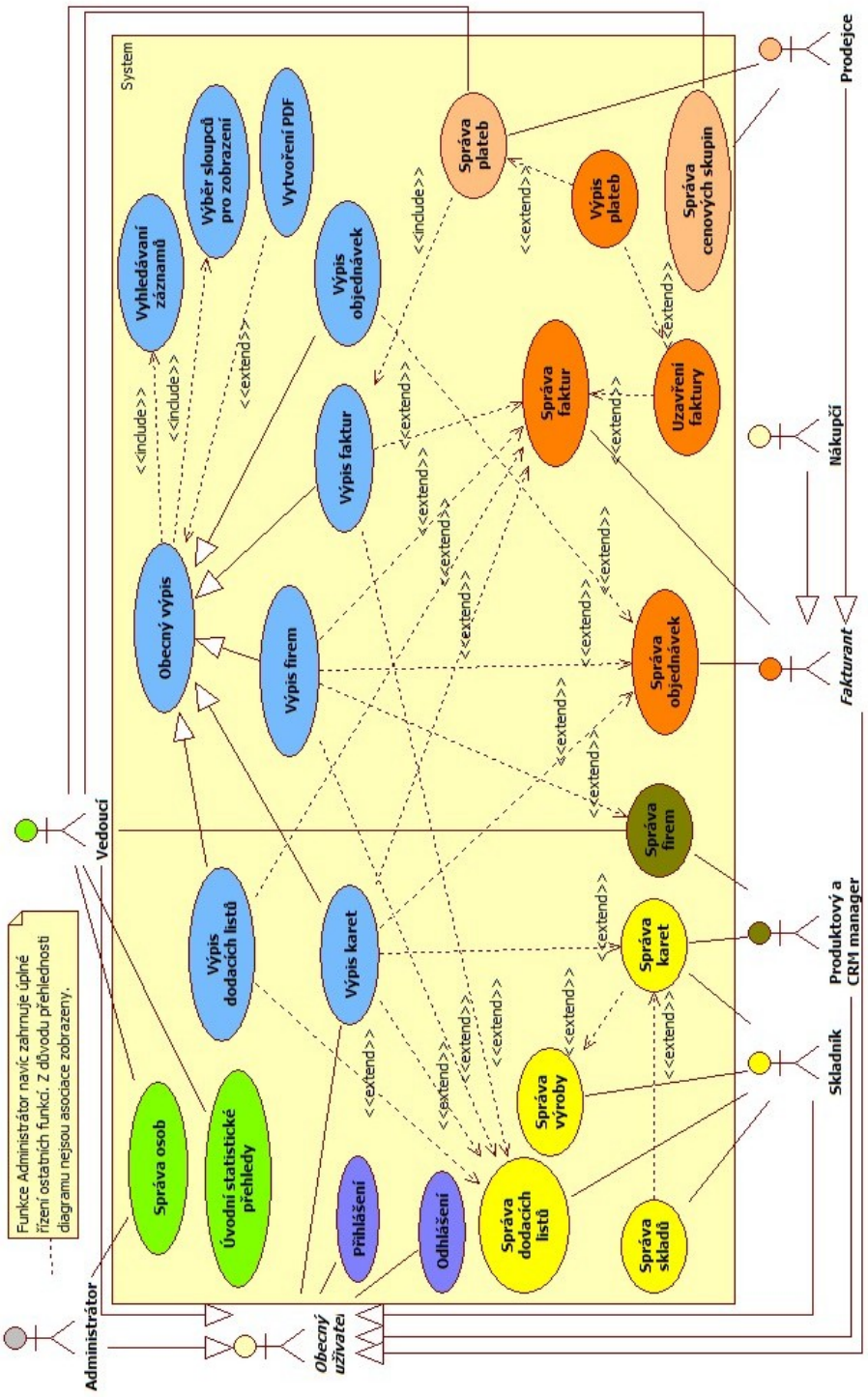
Hlavní případ užití popisuje základních funkce systému zmíněné výše a poskytuje tak jeho celkový přehled. Diagram také zobrazuje interakce aktérů s jejich specifickými funkcemi, čímž vizuálně určuje rozsah působnosti aktérů v systému. Pro zajištění přehlednosti diagram zahrnuje dva abstraktní aktéry, kteří shrnují funkce pro zdědění. Aktér obecný uživatel provádí funkce společné pro všechny aktéry, aktér fakturant zobrazuje abstrakci funkcí společných pro role nákupčí a prodejce. Z důvodu obecné podoby funkcí se diagram nezabývá jejich rozpisem na přijaté a vydané doklady. Samotný diagram je zobrazen na obrázku 2.

3.6.2 Správa karet

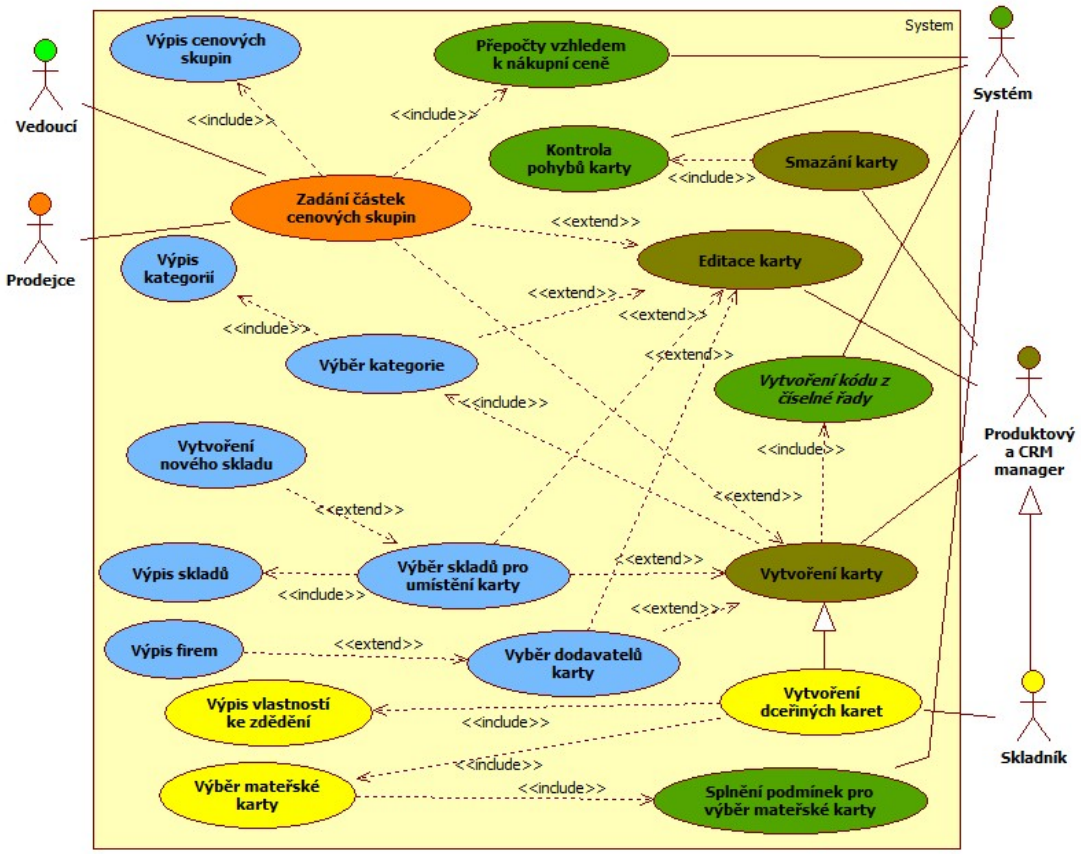
Případ užití správa karet zobrazený na obrázku 3 nám ukazuje, že přístup k základním funkcím pro vytvoření, editaci a mazání karty má aktér produktový a CRM manager. Přístupy k funkcím od něj dědí aktér skladník, který dále využívá vlastní funkci pro vytvoření několika dceřiných karet. Vytvoření dceřiné karty zahrnuje výběr karty mateřské a výpis vlastností, které lze zdědit od mateřské karty. Ostatní funkce jsou identické s funkcí pro vytvoření karty. Na této funkci se podílí aktér systém, který zajistí obsluhu pro splnění podmínek výběru mateřské karty. Systém dále odpovídá za přepočty u zadaných částek cenových skupin, které provádí aktéři vedoucí a prodejce. Systém zajišťuje přiřazení číselného kódu karty, pouze při jejím vytvoření.

3.6.3 Správa výroby

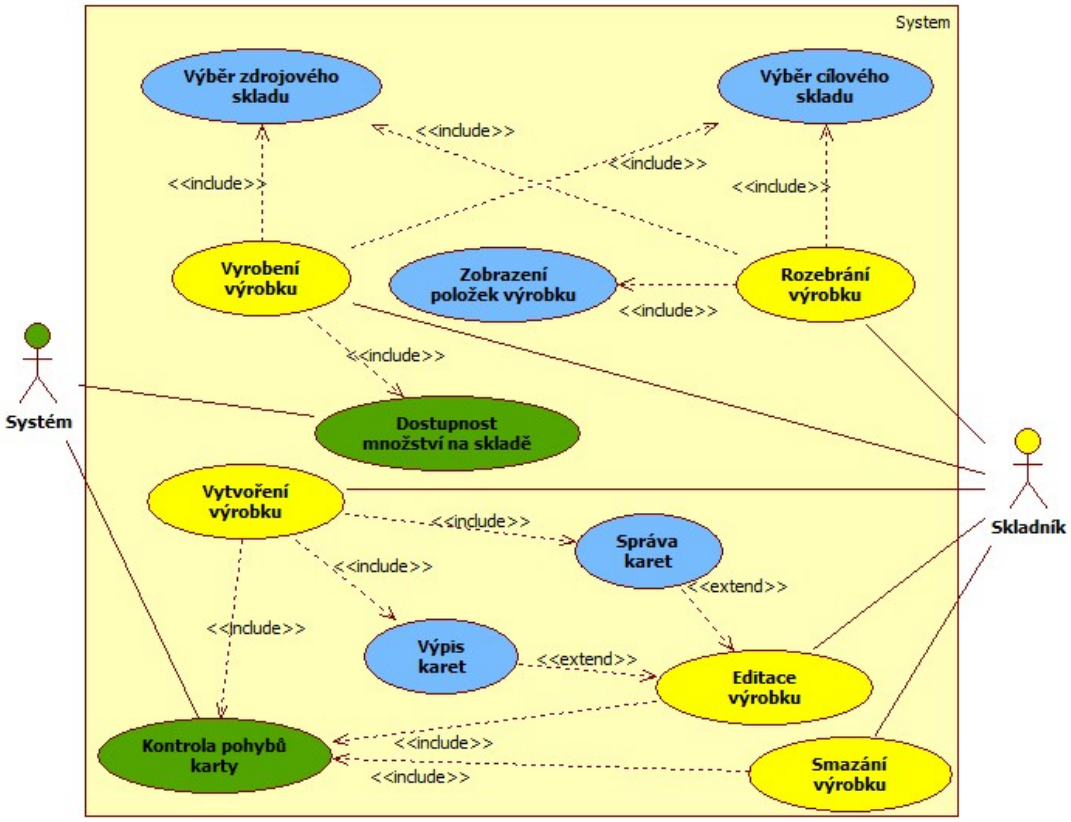
Diagram případu užití na obrázku 4 zachycuje dvě hlavní funkce, vyrobení a rozebrání výrobku, aplikované na výběru zdrojového a cílového skladu. Další funkcí, kterou aktér skladník vykonává, je samotné vytvoření výrobku, bez kterého výrobu nelze uskutečnit.



Obrázek 2: Hlavní případ užití



Obrázek 3: Případ užití Správa karet



Obrázek 4: Případ užití Správa výroby

Vytvoření výrobku se skládá z výběru karet, které určují jeho složení. Aktér systém má odpovědnost za funkci kontroly pohybů výrobku, která ovlivňuje dostupnost jeho základních funkcí. Dále odpovídá za kontrolu množství karet na skladě při funkci vyrobení.

3.6.4 Správa faktur

Další důležitý případ užití znázorňuje obrázek 5. Diagram podrobněji zachycuje obecný přehled funkcí správy faktury, který v této úrovni nerozlišuje mezi přijatou a vydanou fakturou. Proto se pro přístup k funkcím využívá abstraktní aktér fakturant.

Základní operace pro evidenci doplňují funkce typu dobropisování, stornování a především vytvoření faktury z dodacího listu, která urychluje správu byznys procesu. Klíčovou funkci diagramu představuje vytvoření faktury. Aktér systém, v případě přijaté faktury, provádí automatické rozpočítávání nákladů a ocenění karet po uložení faktury. V případě vytvoření faktury vydané, systém odpovídá za funkci automatického rozpočítání slev mezi karty faktury.

3.7 Aktivitní diagramy

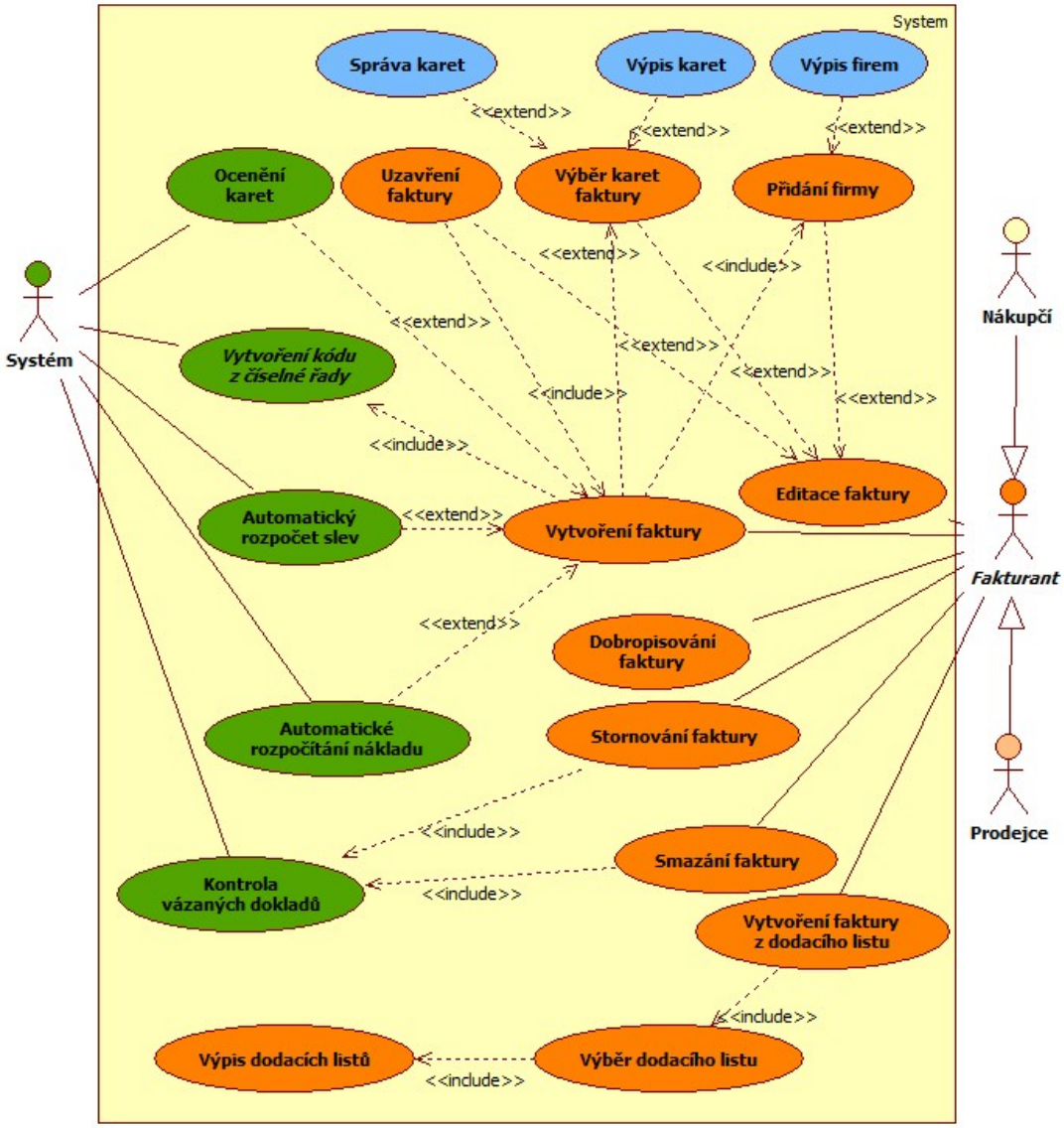
Pro popis procedurální logiky, zachycení byznys procesů a zobrazení toků práce aktérů do procesů vstupujících. V této části se zaměřím na ukázku hlavního procesu prodeje a zachycení klíčových procesů, které specifikují případy užití výše zmíněné. Mezi vybrané procesy patří Vytvoření karty, Vyrobení výrobku a Vytvoření faktury z dodacího listu.

3.7.1 Proces prodeje

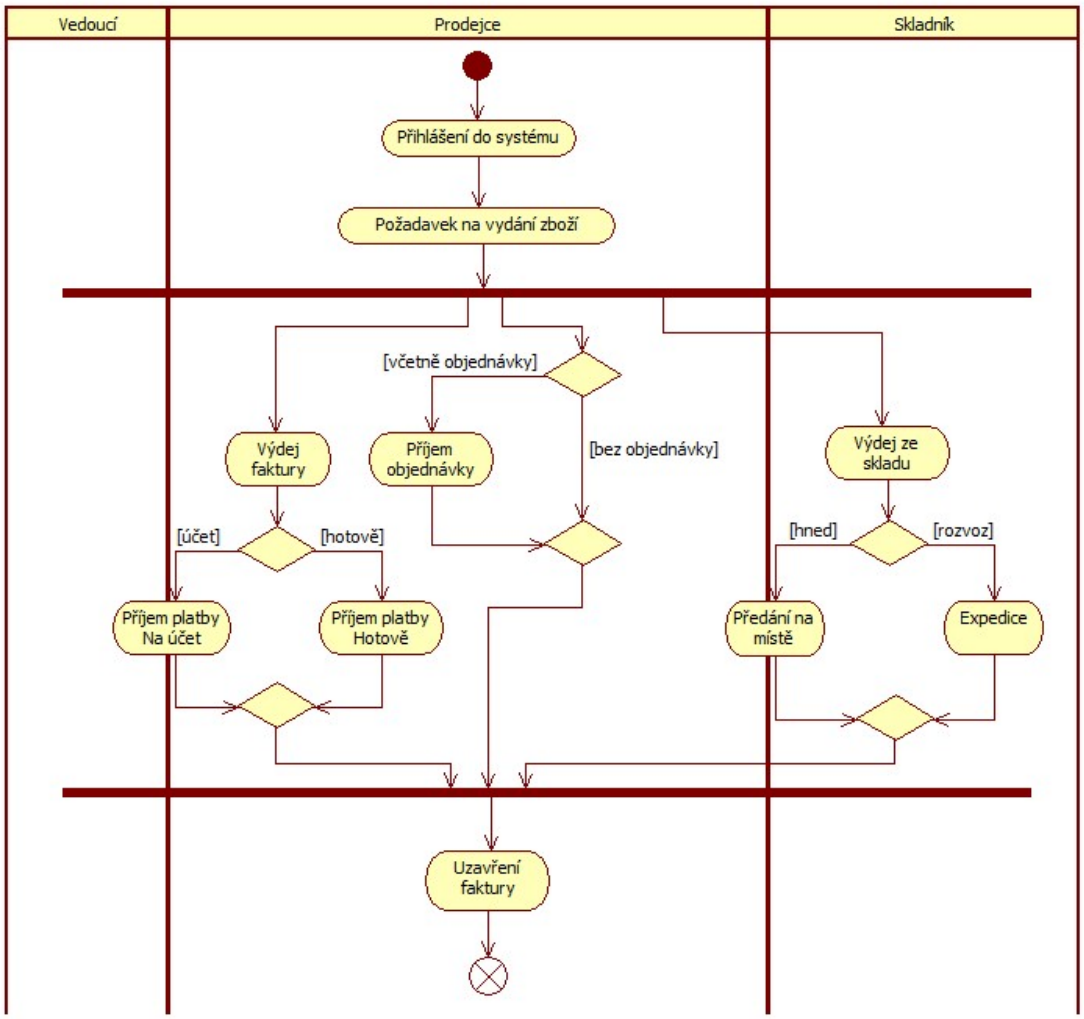
Obrázek 6 zobrazuje aktivitní diagram určený k popisu základního procesu prodeje ve firmě. Z diagramu je zřejmé, že hlavním dokladem procesu je faktura a dodací list. Příjem objednávky není v procesu prodeje povinný. Zpracování dokladů probíhá nezávisle na pořadí, a proto je součástí paralelního bloku. Hlavní odpovědnost za proces přebírá aktér prodejce, který odpovídá za výdej faktury, příjem objednávky a podílí se na předání zboží a příjmu platby. O příjem platby na účet se stará vedoucí, který poskytuje přístup k přehledu plateb. Standardní výdej ze skladu provádí aktér skladník, který předá zboží na místě. Dále může provádět expedici zboží, která ovšem není součástí zadání práce. Celkové uzavření procesu souvisí s uzavřením dokladu faktury.

3.7.2 Vytvoření faktury převodem

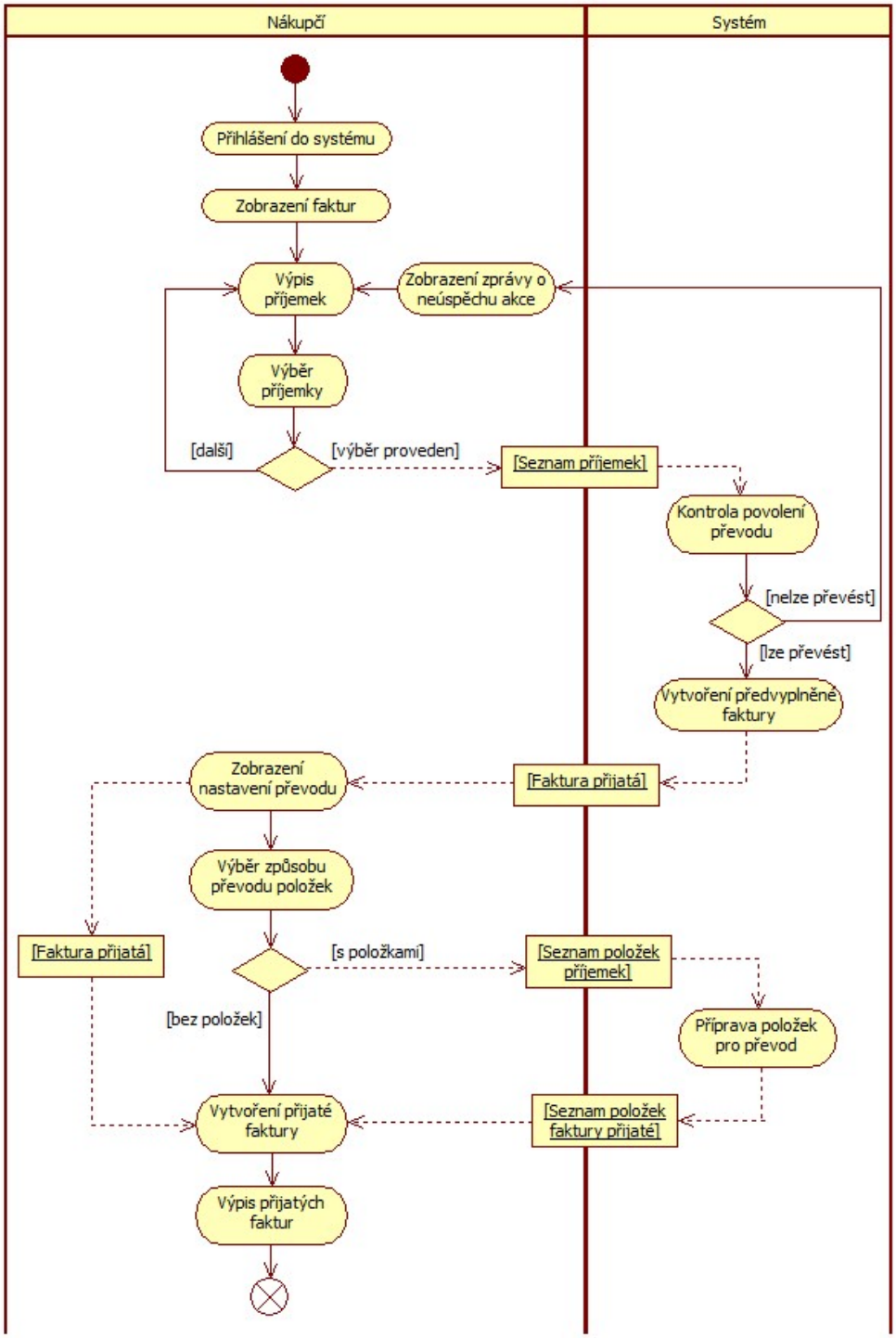
Aktivitní diagram na obrázku 7 prezentuje toky činností procesu vytvoření faktury přijaté z příjemky, který zaznamená sledovatelnou vazbu mezi doklady. Proces určuje aktér nákupčí a skládá se z výběru jedné i několika příjemek, kde, po splnění kontrol systémem,



Obrázek 5: Případ užití Správa faktur



Obrázek 6: Aktivitní diagram Hlavní proces prodeje



Obrázek 7: Aktivitní diagram Vytvoření faktury z dodacího listu

dochází k výběru způsobu převodu - s položkami či bez nich. Systém poté připraví předvyplněnou fakturu nad společnými daty z příjemek se všemi položkami či bez nich.

Aktivitní diagram popisuje principiálně stejný proces pro všechny směry dokladů. Jedná se tedy o převod faktury přijaté/vydané na dodací list typu příjemka/výdejka, a naopak.

3.7.3 Vytvoření karty

Aktivitní diagram na obrázku 8 ukazuje toky činností při procesu vytvoření karty do systému. Za základní vytvoření zodpovídají aktéři skladník a produktový manager. Výběr kategorie, skladů a dodavatelů karty lze provádět nezávisle na pořadí. Na vytvoření karty se také podílí aktér prodejce, který určí částky prodejních cen nad vybranými skupinami.

3.7.4 Výroba

Princip výroby zakládá na umožnění uživateli seskupení položek pod kartu typu výrobek a provádění jeho naskladnění a vyskladnění. Vyskladnění lze spojit s funkcí rozebrání výrobku, ovšem za naskladnění výrobku zodpovídá proces vyrobení výrobku, který popisuje aktivitní diagram níže.

Obrázek 9 tedy prezentuje aktivitní diagram určený k popisu procesu vyrobení výrobku nad systémem. Výroba spadá pod aktéra skladník, který zodpovídá za výběr či vytvoření výrobku a určuje sklady zdrojové a cílové pro vyrobení. Systém se stará o povolení vyrobení, snížení množství na zdrojovém skladu a navýšení množství na cílovém skladu.

3.8 Scénáře Use-Case modelů

Případy užití specifikovaly hlavní funkce systému. Pro podrobnější popis jednotlivých funkcí v use-case modelech slouží scénáře případů užití. Scénáře jsou podrobně popisovány minispecifikacemi. Pro ukázku se popisuje funkce vytvoření faktury, další popsané funkce jsou v sekci A.

3.8.1 Vytvoření přijaté faktury

Záměr: Nákupčí potřebuje uložit přijatou fakturu do systému

Rozsah: Systém BIGGIE

Úroveň: Uživatelský cíl

Primární aktor: Nákupčí

Účastnící a zájmy: Nákupčí požaduje vložení přijaté faktury do systému a ovlivnění nákupních cen karet faktury v systému

Vstupní podmínka: Aktor je přihlášen v systému

Minimální záruky: Aktor je systémem informován o neúspěchu uložení a může proces opakovat bez ztráty vyplněných dat

Záruky úspěchu: Nákupčí je systémem informován o úspěšném vložení a výpis karet obsahuje přepočtené ceny

Spouštěč: Nákupčí má potřebu evidovat přijatou fakturu

Hlavní scénář:

1. Nákupčí zadá obecné informace do formuláře.
2. Nákupčí požaduje evidování firmy.
3. Systém nákupčímu poskytne výpis firem.
4. Nákupčí vybere firmu.
5. Nákupčí akceptuje nastavení zaokrouhlování dokladu.
6. Nákupčí přidává položky na fakturu.
7. Systém nákupčímu poskytne výpis karet.
8. Nákupčí vybere karty ze seznamu.
9. Systém provede kontrolu, zda lze karty přidat na fakturu.
10. Systém vloží vybrané karty na vytvářený doklad.
11. Nákupčí ukončil výběr karet.
12. Nákupčí provede úpravu údajů načtených karet.
13. Nákupčí vloží fakturu přijatou.
14. Systém provede validaci dat faktury.
15. Systém provede Vytvoření kódu z číselné řady.
16. Systém provede zároveň uložení faktury, položek faktury a Ocenění karet.
17. Systém informuje studenta o úspěšném uložení faktury.
18. Nákupčí zkontroluje nový záznam v seznamu faktur přijatých.

Rozšíření:

1–13a. Nákupčí může kdykoliv ukončit vytvoření přijaté faktury.
1–18a. Systém ukončí případ užití bez uložení jakýchkoliv údajů.

4a. Firma není v seznamu.
4a1. Nákupčí vyplní ručně údaje o firmě.

5a. Nákupčí požaduje změnu výchozího zaokrouhlování faktury.
5a1. Systém poskytne nákupčímu nastavení zaokrouhlování platné pro aktuální fakturu.
5a2. Nákupčí vybere nastavení zakokrouhlování.

6a. Nákupčí požaduje zadání nesystémové karty.
6a1. Systém vloží do seznamu nesystémovou položku (pokračuje

krokem 11).

6b. Nákupčí požaduje zadání nákladu.
6b1. Nákupčí vloží náklad s cenou do faktury.
6b2. Systém provede Automatické rozpočítání nákladu.
6b3. Systém vloží do seznamu položku bez vazby na kartu (pokračuje krokem 11).

6c. Nákupčí zadá vyhledání karty přes systémový kód.
6c1. Systém nalezne kartu (pokračuje krokem 11).

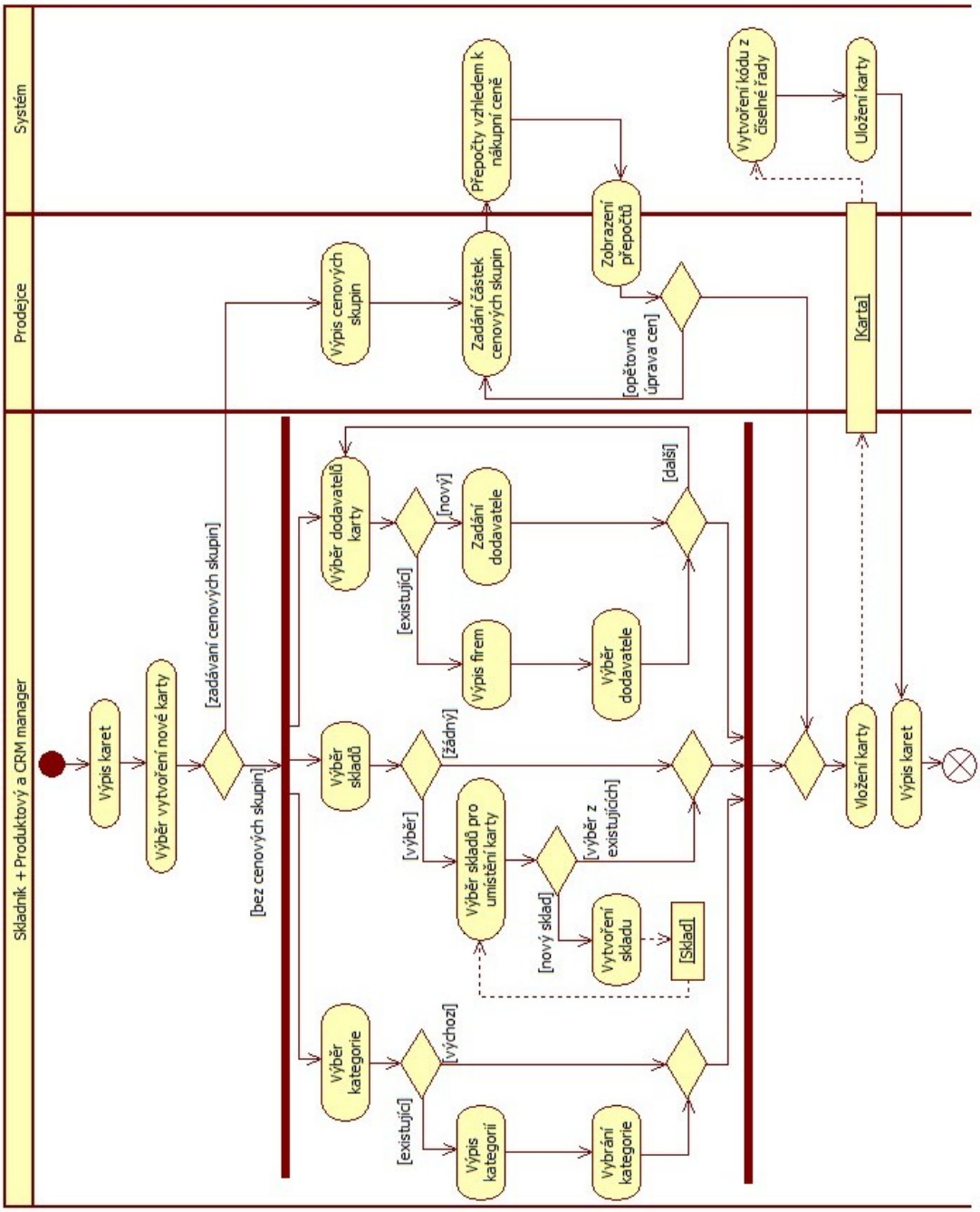
6c1b. Systém nenalezl kartu.
6c1b. Systém informuje nákupčího o nenalezení karty (pokračuje krokem 6).

9a. Systém ověřil nemožnost přidání karty na fakturu.
9a1. Systém informuje nákupčího o nemožnosti přidání karty (pokračuje krokem 6).

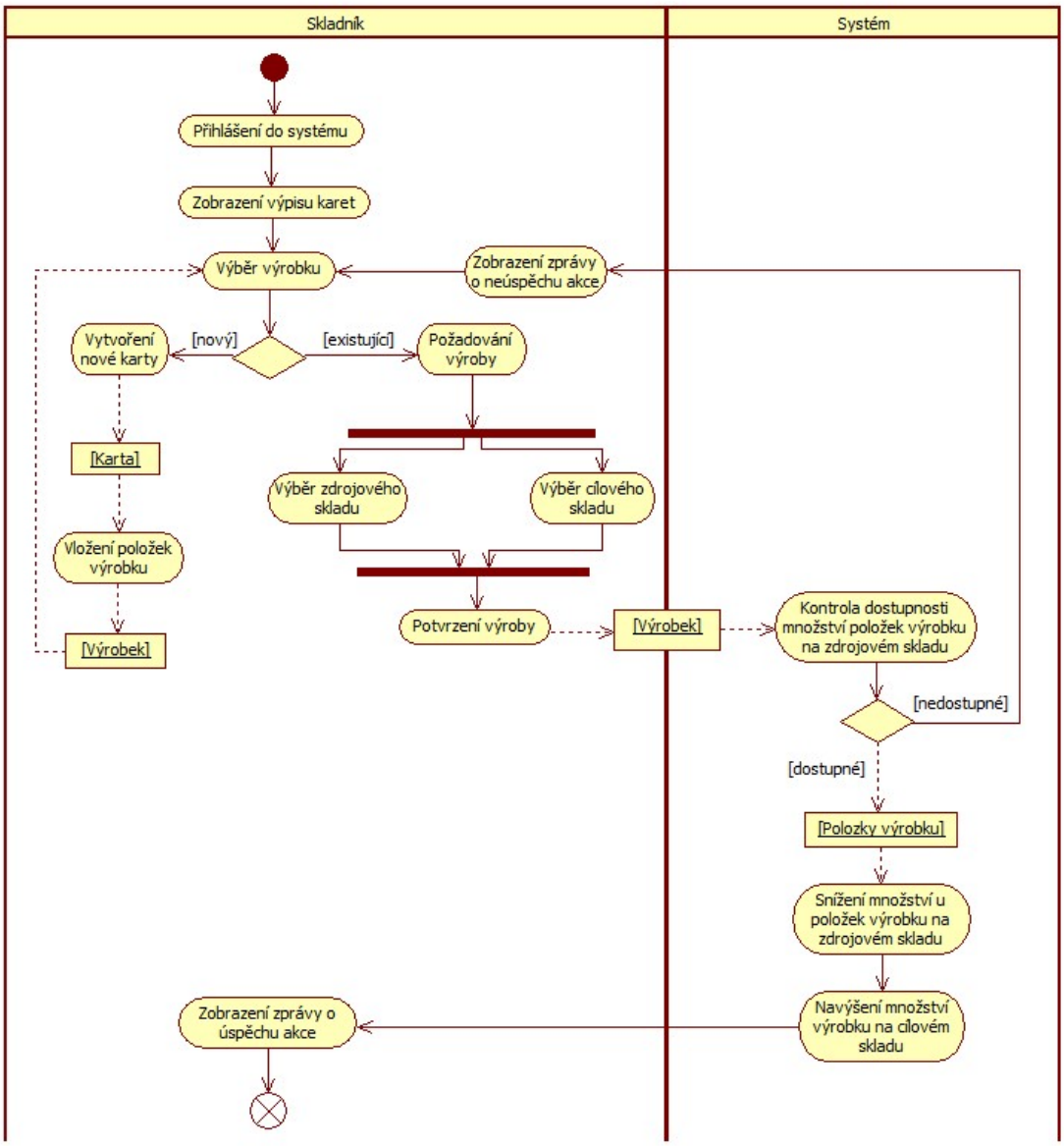
11a. Nákupčí pokračuje v přidávání karet a pokračuje krokem 6.

14a. Systém provedl validaci dat s neúspěchem.
14a1. Systém informuje nákupčího zprávou s výsledkem validace.
14a2. Nákupčí provede úpravy a vrací se ke kroku 13.

16a. Systém neprovedl úspěšné uložení všech dat.
16a1. Systém informuje nákupčího o možnosti opětovného uložení faktury (vrací se ke kroku 13).



Obrázek 8: Aktivitní diagram Vytvoření karty



Obrázek 9: Aktivitní diagram Vyrobení výrobku

4 Analýza

Analýza propojuje požadavky na systém s jeho budoucím návrhem implementace z pohledu objektů v systému. Kapitola prezentuje realizaci logické, statické a dynamické struktury systému s využitím technik datové analýzy a UML diagramů.

Stojí za zmínění, že rozsah mnou vypracovaných tabulek tvoří základní strukturu databáze produktu XEVOS ERP Live, ale realizace databázové struktury bude prezentována menším počtem tabulek užitých nad aktuálním systém BIGGIE. Jen pro představu, návrh databázové struktury XEVOS ERP Live přesáhl hodnotu padesáti tabulek.

4.1 Třídní diagram

Třídní diagram je použit k popisu typů objektů v systému a statických vztahů, díky kterým jsou objekty spojovány. Statickou strukturu a logiku systému, zachycenou pomocí mezi sebou komunikujících tříd, ukazuje obrázek 10.

Z důvodu přehlednosti zachycuje zobrazený třídní diagram komunikaci podstatných tříd systému. Ostatní třídy figurují jako datové typy a pouze se odkazují na třídy v diagramu, tedy nemají další relevantní vztahy. Zároveň diagram určuje rozdělení tříd do balíčků. Balíček pro zakázku ukazuje realizovaný požadavek na nezávislost dokladů oproti ostatním třídám.

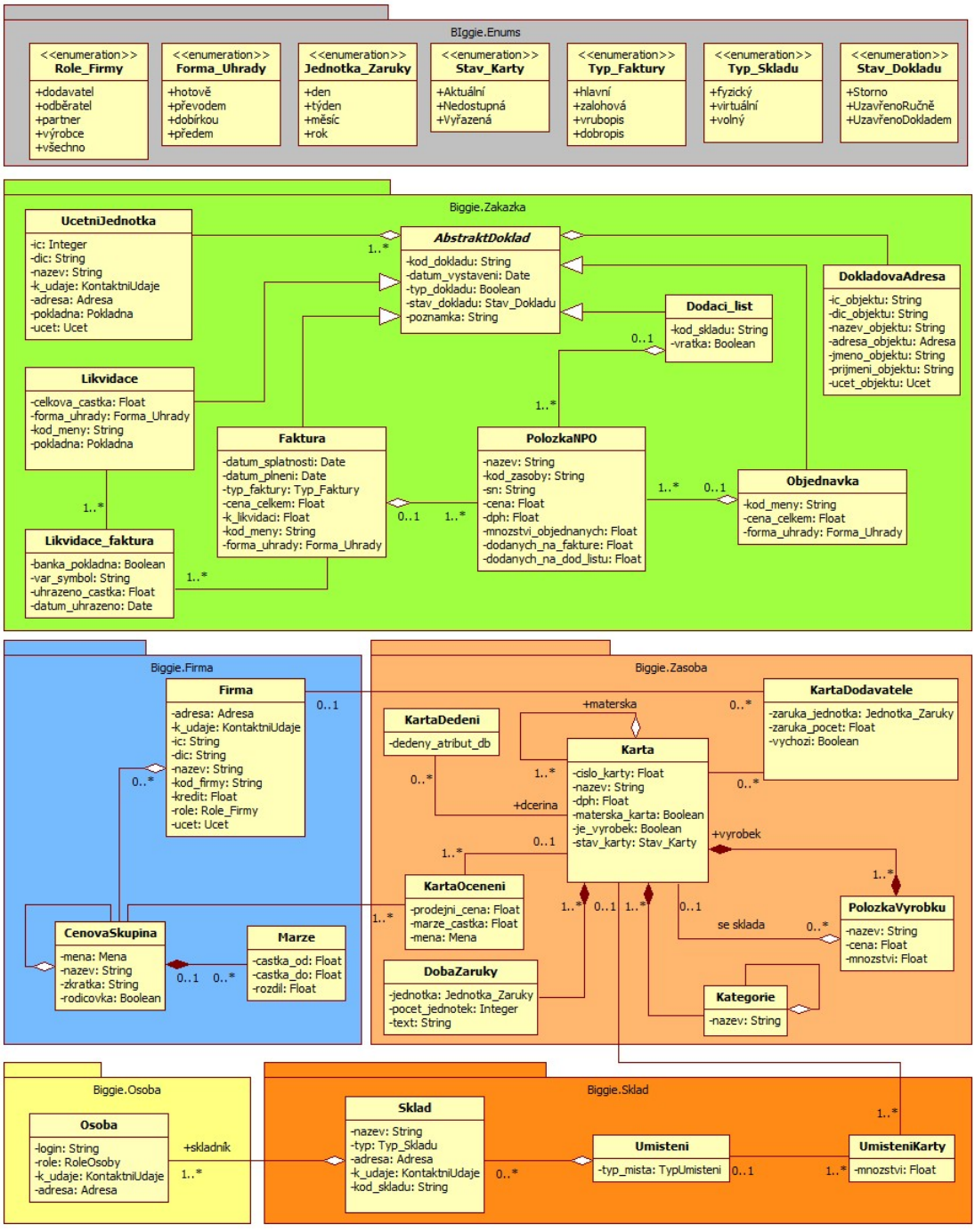
Pro představu všech navrhnutých balíčků poskytuji pouze jejich výpis - `Zakazka`, `Sklad`, `Zasoba`, `Firma`, `Osoba`, `Finance`, `Kontakt`, `CiselneRady`, `Obecne`, `Enums`.

4.2 Datová analýza

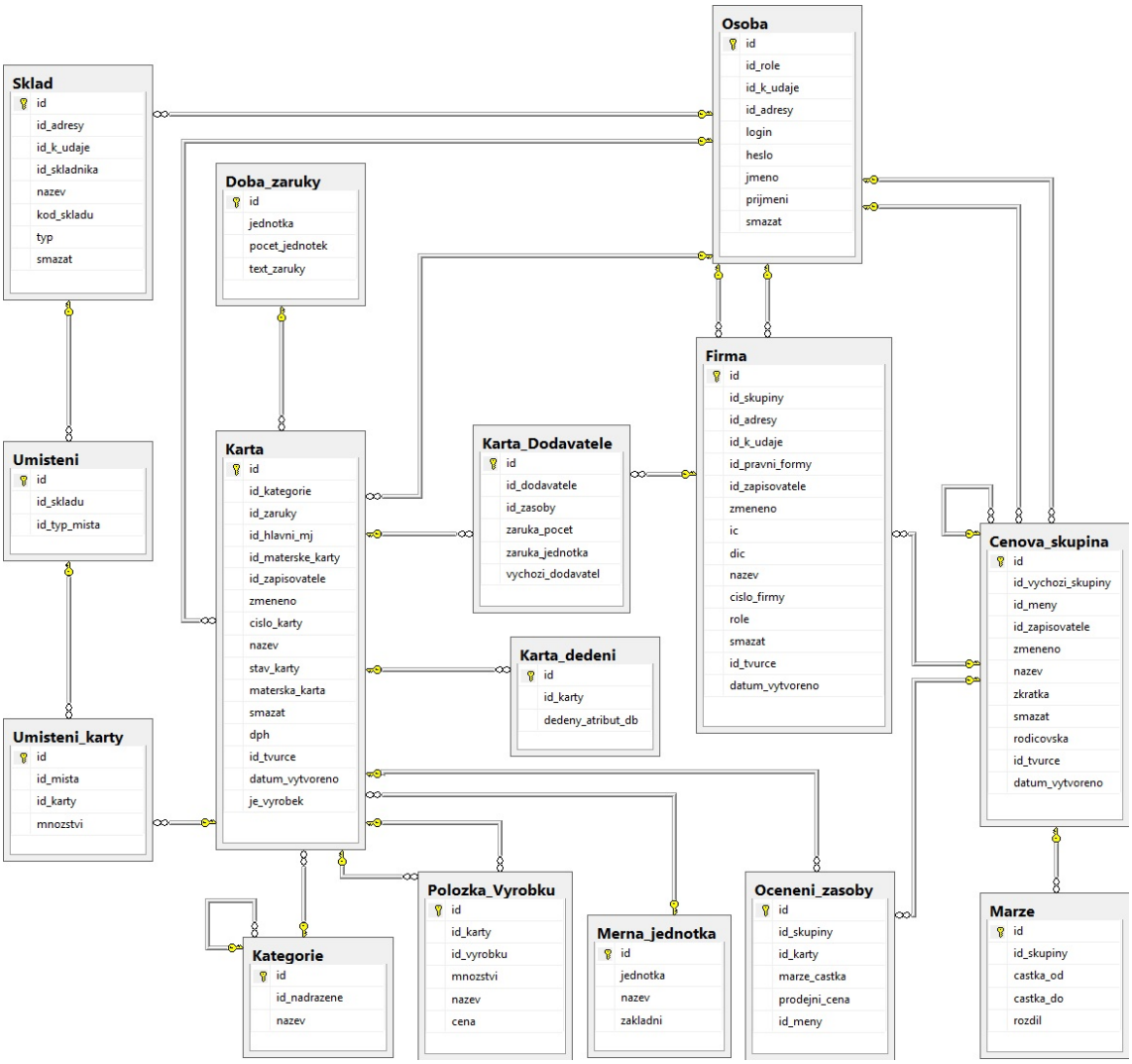
Důležitou částí návrhu databázového systému byla volba vhodného uložení dat, definice tabulek a jejich atributů, včetně určení vazebních vztahů mezi nimi. Datová analýza upřesňuje třídní diagram na obrázku 10, a to za pomocí ER diagramu, lineárního zápisu a datového slovníku.

4.2.1 ER diagram

Vzhledem k tomu, že celkový ER diagram obsahuje velké množství tabulek, nebyl zahrnut do této části dokumentu. Další rozsáhlou oblastí jsou entity určené pro doklady, jejich ER diagram utvořil samostatnou část zobrazenou na obrázku 20 v příloze B na konci dokumentu. Dále tedy popisuji datovou sturkturu nad entitou karta a entitami s ní spojených.



Obrázek 10: Třídní diagram nad hlavními objekty



Obrázek 11: Návrhový ER diagram pro entitu Karta

Návrhový ER diagram na obrázku 11 zobrazuje vazební vztahy entity karta. Celkově jde o obecný popis struktury databáze bez zahrnutí entit určených pro doklady, tedy bez balíčku zakázka, který ukazuje třídní diagram výše na obrázku 10.

Lineární zápis typů entit

[**primary key**] [foreign key]

Doba_zaruky(**id**, jednotka, pocet_jednotek, text_zaruky)
Karta(**id**, id_kategorie, id_zaruky, id_hlavni_mj, id_materske_karty, id_zapisovatele, id_tvurce, datum_vytvoreno, zmeneno, cislo_karty, nazev, stav_karty, materska_karta, dph, je_vyrobek, smazat)
Karta_dedeni(**id**, id_karty, dedeny_atribut_db)
Kategorie(**id**, id_nadrazene, nazev)
Merna_jednotka(**id**, jednotka, nazev, zakladni)
Sklad(**id**, id_adresy, id_k_udaje, id_skladnika, nazev, kod_skladu, typ, smazat)
Umisteni(**id**, id_skladu, id_typ_mista
Osoba(**id**, id_role, id_adresy, id_k_udaje, login, heslo, jmeno, prijmeni, smazat)
Firma(**id**, id_skupiny, id_adresy, id_k_udaje, id_pravni_formy, id_zapisovatele, id_tvurce, datum_vytvoreno, zmeneno, cislo_firmy, nazev, ic, dic, role, smazat)
Cenova_skupina(**id**, id_meny, id_vychozi_skupiny, id_zapisovatele, id_tvurce, datum_vytvoreno, zmeneno, nazev, zkratka, rodicovska, smazat)
Marze(**id**, id_skupiny, castka_od, castka_do, rozdil)
Karta_Dodavatele(**id**, id_dodavatele, id_zasoby, zaruka_pocet, zaruka_jednotka, vychozi_dodavatel)
Umisteni_karty(**id**, id_mista, id_karty, mnozstvi)
Polozka_Vyrobku(**id**, id_karty, id_vyrobku, mnozstvi, cena, nazev)
Oceneni_zasoby(**id**, id_skupiny, id_karty, id_meny, marze_castka, prodejni_cena)

4.2.2 Datový slovník

Popis účelů tabulek reprezentujících entity ER diagramu:

- **Doba_zaruky** - uchovává číselník záruk.
- **Karta** - významná entita ER diagramu, která uchovává informace o kartě a odkazy na další entity ji upřesňující.
- **Karta_dedeni** - uchovává výběr děděných atributů u dceřiné karty od karty mateřské.
- **Kategorie** - číselník na sebe odkazujících se kategorií.
- **Merna_jednotka** - číselník definovaných měrných jednotek.
- **Sklad** - významná entita ER diagramu, která uchovává informace o skladu a odkazy na další entity jej upřesňující.

- **Umisteni** - upřesňuje několik druhů umístění na skladě podle typu umístění.
- **Osoba** - uchovává osoby do systému vstupujících a v případě osoby jako uživatele systému se evidují jeho změny nad vybranými záznamy. V zobrazeném ER diagramu jde o entity karta, sklad a firma.
- **Firma** - významná entita ER diagramu, která uchovává informace o firmě a odkazy na další entity ji upřesňující.
- **Cenova_skupina** - definuje číselník skupin, které se mohou odkazovat na rodičovskou (výchozí) skupinu.
- **Marže** - uchovává zadané rozsahy částek skupin a k nim určené rozdílové hodnoty.

Tab Karta					
Atribut	**Null**	**Typ**	**Klíč**	**Poznámka**	**Index**
id	NE	int	PK	inkrementální číslo	ANO
id_kategorie	NE	int	FK	z tab Kategorie	ANO
id_zaruky	NE	int	FK	z tab Doba_zaruky	ANO
id_hlavni_mj	NE	int	FK	z tab Merna_jednotka	ANO
id_materske_karty	ANO	int	FK	z tab Karta - mateřská karta	ANO
id_zapisovatele	NE	int	FK	z tab Osoba - zapisovatel	ANO
id_tvurce	NE	int	FK	z tab Osoba - tvůrce záznamu	ANO
datum_vytvoreno	NE	datetime	FK	datum vytvoření záznamu	NE
zmeneno	NE	datetime	NE	časový údaj o změně	NE
cislo_karty	NE	varchar	NE	číslo přiřazené z číselné řady	ANO
nazev	NE	varchar	NE	název karty	ANO
stav_karty	NE	tinyint	NE	název karty	NE
materska_karta	ANO	bit	NE	indikace mateřské karty (true)	NE
dph	NE	float	NE	hodnota DPH karty	NE
je_vyrobek	NE	bit	NE	indikace výrobku (true)	NE
smazat	NE	bit	NE	zneplatněný záznam (true)	NE

Tabulka 2: Popis entity Karta

Popis účelů tabulek reprezentujících vazební tabulky mezi entitami ER diagramu:

- **Umisteni_karty** - sleduje počet karet na umístění skladu.
- **Polozka_Vyrobku** - uchovává výběr položek karty typu výrobek.
- **Oceneni_zasoby** - uchovává zadané částky cenových skupin pod kartou.
- **Karta_Dodavatele** - uchovává dodavatelé karty.

Vzorovou ukázku datového slovníku nabízí tabulka 2 zobrazující atributy databázové tabulky Karta, jakožto hlavního objektu prezentované části. Ostatní tabulky s popisem atributů navrhované databáze přikládám v příloze práce na kompaktním disku.

4.2.3 Využití uložených procedur a pohledů

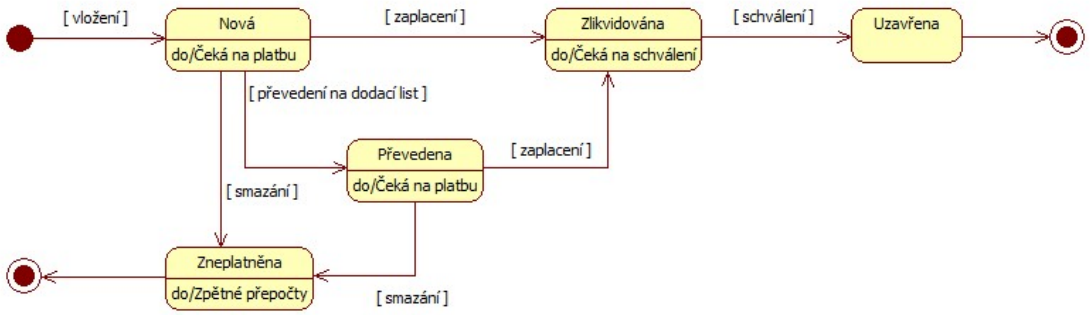
Uložená procedura (Stored procedure) představuje databázový objekt, který neobsahuje data, ale část programu, který se nad daty v databázi má vykonávat, a který je nezávislý od svého okolí. Nad databází SQL Server došlo k návrhu uložených procedur, které uchovávají databázové dotazy pro vkládání a editaci záznamů, a to nad všemi tabulkami. Názvy procedur mají daný vzor `sp[Nazev_tabulky][Nazev_operace]`.

Pohled (View) představuje databázový objekt, který uživateli poskytuje data ve stejné podobě jako tabulka. Ovšem pohled obsahuje pouze předpis, jakým způsobem mají být data získána z tabulek či jiných pohledů. Práce s pohledem odpovídá pravidlům při prací nad tabulkou. Navržení pohledů nad klíčovými objekty v systému umožnilo ucelený přístup k jejich atributům. Pro názornost byly vytvořeny tyto pohledy - `ViewKarta`, `ViewSklad`, `ViewPrijemky`, `ViewVydejky` a další.

Jak uložené procedury, tak pohledy, musí být při změně atributů v databázi aktualizovány, aby se zajistila správná funkčnost. Pro uložené procedury typu elementárních operací nad záznamy existují generátory, které využívají k jejich sestavení neměnnou formu dotazů. Informace o tabulkách získávají dotazy nad strukturou databáze.

4.3 Stavový diagram entity Faktura

Stavový diagram zobrazuje životní cyklus objektu, události způsobující přechody z jednoho stavu do jiného a akce, které vyplývají z této změny stavu [6].



Obrázek 12: Stavový diagram entity Faktura

Diagram na obrázku 12 zobrazuje stavy, události a akce faktury jako hlavního objektu byznys procesu.

5 Návrh

Kapitola upřesňuje analýzu nad skutečným implementačním prostředím. Probírá mapování analýzou navržených softwarových komponent na architekturu systému s využitím diagramů komponent a návrhových vzorů.

5.1 Prostředí

Aplikace staví na .NET frameworku 3.5 s podporou jazyka C#, kde s potřebou tvorby systému jako webové aplikace, se využívá technologie ASP.NET Web Forms. Provádění asynchronní komunikace mezi požadavky zajišťuje knihovna AJAX Extensions rozšířená o připravené ovládací prvky v balíčku Ajax Control Toolkit. Dále se do návrhu prostředí zahrnují zkušební verze komponent Infragistics .NET Advantages a Telerik, které nabízejí hotové ovládací prvky, vybudované na výše zmíněných technologií. Řízení dat se provádí nad relačním databázovým systémem SQL Server.

Poznámka 5.1 V počáteční době vývoje systému se pracovalo nad .NET frameworkem 3.0, což ovlivnilo jeho celý další návrh. Nemohly být tedy zahrnuty dnes již osvědčené klíčové prvky verze 3.5 frameworku .NET, zejména technologie pro dotazování a zpracování dat LINQ (Language Integrated Query). Tím, že aplikace staví na .NET frameworku 3.5 se myslí to, že využívá aktualizované knihovny této verze.

5.1.1 Základní ovládací prvky systému

Rozsáhlost použitých ovládacích prvků ASP.NET a jejich vlastností umožňuje prezentování pouze těch nejdůležitějších, nad kterými bude základ systému vybudován. Jedná se především o prvky zobrazující výpisy záznamů nad jednotlivými agendami (podagendami), které jsou sepsány dle důležitosti umístění na stránce:

- `TabContainer`
  Stežejní prvek celého návrhu vzhledu systému, který realizoval požadavek na vícezáložkový systém. Jedná se o AJAXový prvek knihovny Control Toolkit, jenž obsahuje kolekci jednotlivých záložek typu `TabPanel`.

- `TabPanel`
  Obsahuje vlastnosti jako `HeaderText`, `HeaderTemplate`, kde se pro nastavení obsahu záložky užívá `ContentTemplate`. Jedna záložka reprezentuje jednu podagendu systému (faktury přijaté, výdejky, karty, atd.).

- `GridView / RadGrid / UltraWebGrid`
  Všechny prvky obsahují stejnou základní funkčnost definovanou ASP.NET prvkem `GridView`. `RadGrid` a `UltraWebGrid` jsou pouze prvky doplněné o funkce podle jejich implementace - Telerik, Infragistics. Jedná se o datové ovládací prvky, jejichž účelem je zobrazování rozsáhlých objemů dat. Podporují vyspělé schopnosti, kde

z nich využity budou šablony, řazení nebo stránkování. Šablony umožní realizaci funkce pro vygenerování sloupců gridu podle uživatelova výběru.

- `ObjectDataSource`
  Patří mezi ovládací prvky pro zdroje dat, které umožňují se deklarativně vázat na různé typy zdrojů dat bez nutnosti vytváření složitých kódů. `ObjectDataSource` umožňuje vytvořit deklarativní spojení mezi webovými ovládacími prvky a komponentou pro přístup k datům. Komponenta poskytuje metody pro dotazování dat a jejich aktualizaci. V aplikaci se `ObjectDataSource` bude nejčastěji využívat s datovým ovládacím prvkem typu grid.

- `TreeView / RadTreeView / UltraWebTree`
  Ovládací prvek `TreeView` technologie ASP.NET zobrazuje jen hierarchická data a lze jej pouze připojit k ovládacím prvkům pro zdroje dat - `XmlDataSource` a `SiteMapDataSource`. Stejně tak prvek `UltraWebTree` od Intfagistic. Naopak Telerik komponentu `RadTreeView` lze připojit ke všem ovládacím prvků pro zdroje dat. V systému bude napojen na zdroj dat typu `ObjectDataSource` a použit pro zobrazení stromu kategorií.

5.1.2 Dynamický vs. statický kód stránky

Nývrh aplikace provázelo počáteční rozhodnutí jakým stylem implementovat kód pro vzhled stránky. Na základě nalezených informací a shrnutí požadavků, došlo v návrhu k částečnému upřednostnění dynamických stránek před statickými. Statický kód aspx stránky obsahuje části nutné pro její správný chod či neměnné prvky, ovšem co v ní bude zobrazeno určuje až dynamický kód.

Příklady obsahu statické části stránky - hlavní menu, připojení referencí na JavaScriptové knihovny nebo vytvoření funkcí, přidání ovládacích prvků typu `ScriptManager`, `UpdatePanel` a `Panel`, do kterého se bude zobrazovat obsah.

Během práce nad systémem a jeho návrhu se projevily následující výhody dynamického kódu, které byly klíčové při rozhodování:

- **Podpora objektově orientovaného programování**
  Vytvořením podpůrných třídních komponent a abstrakcí kódu tříd, došlo k nahrazení, jednak opakujících se částí stránek, tak nutných postupů tvorby interakce ovládacích prvků.

- **Dynamická změna obsahu**
  Vhodné využití dynamiky se ukázalo právě nad návrhem implementace informačního systému ve webovém prostředí. U stránek se složitou logikou se lépe řídí reakce na načtená data (byznys objekty, data v paměti), což umožňuje měnit její vzhled, prvky, ale i události.

- **Požadavek na změny v návrhu**
  Od vedení firmy bylo požadováno řešení vzhledu stránek takové, které by umožňovalo variabilitu změn použitých ovládacích prvků či rozmístění. Vytváření stránek dynamicky splňuje takové požadavky s minimálním kódem navíc, což u statických stránek bývá problém vzhledem k dané funkčnosti ASP.NET.

- **Přehlednost aktuálního stavu prvků**
  Člověk mající potřebné znalosti základních technik ASP.NET využívá dynamický kód ke sledování stavů ovládacích prvků, podrobným procházením příkazů a dostává možnost jejich řízení.

Z podstaty každého srovnání dvou technik vyplynuly tyto nevýhody, které především pramení z nutnosti znalosti ovládacích prvků na programové úrovni:

- **Nutné zkušenosti s ovládacími prvky a procesem vytváření stránky**
  Během vývoje může docházet k projevování nestandardních chyb i chyb základních způsobených přesnou neznalostí ovládacích prvků a procesu vytvoření stránky.

- **Nevyužití toolboxu a průvodců nastavování ovládacích prvků**
  Odpadá jedna ze základních výhod ASP.NET, a to přehledné využití ovládacích prvků z toolboxu a podpůrných průvodců nastavování jejich vlastností a interakcí. Programátor musí znát, či spíše v dokumentacích dohledat, vlastnosti a techniky pro jejich řízení v programovém kódu.

- **Statické stránky**
  V případě výskytu vysoce statické webové stránky, bez nutnosti typové kontroly, stačí použít prvek pro zobrazení dat (`ListView`, `GrivView`) a připojit jej ke zdroji dat (např. `SqlDataSource`). Takové stránky ovšem návrh implementace systému nenabízel.

- **Nepřehlednost kódů**
  Pro nezainteresovaného člověka v aplikaci působí velké množství dynamického kódu nepřehledným dojmem, který se umocňuje, pokud nemá potřebné znalosti základních technik ASP.NET.

5.2 Vícevrstvá architektura

Architektura aplikace má tři logické vrstvy, které jsou oddělené. Každá vrstva interaguje pouze s vrstvou přímo pod ní, a má svou zvláštní funkci, za kterou je zodpovědná. To co vrstva dělá je skryto před ostatními vrstvami, což umožňuje změnit nebo aktualizovat jednu vrstvu bez rekompilace nebo změny dalších vrstev. Grafické znázornění navrhnuté vrstvené architektury představuje obrázek 13.

Pohled ze strany MVC

Návrhový vzor Model-View-Controller (MVC) rozděluje modelování aplikace do tří částí:

- Model řídí chování a data aplikační domény, odpovídá na požadavky o stavu od části View a na instrukce ke změnění stavu od části Controller.
- View řídí zobrazení informací.
- Controller reaguje na události pocházející od uživatele a zajišťuje změny v modelu nebo v pohledu.

ASP.NET web formulář se skládá ze souboru stranka.aspx a stranka.aspx.cs. Soubor s příponou aspx představuje HTML kód web formuláře, který může obsahovat tagy pro serverové prvky. V MVC aspx soubor plní funkci View tím, že zajišťuje způsob zobrazení informací. Soubor s příponou aspx.cs obsahuje kódy pro obsluhu událostí stránky aspx v jazyce C#, čímž plní funkci Controlleru.

Část Model představuje aplikační logiku, která se v našem případě skládá z vrstev pro řízení logiky systému (BLL) a řízení přístupu k datům (DAL).

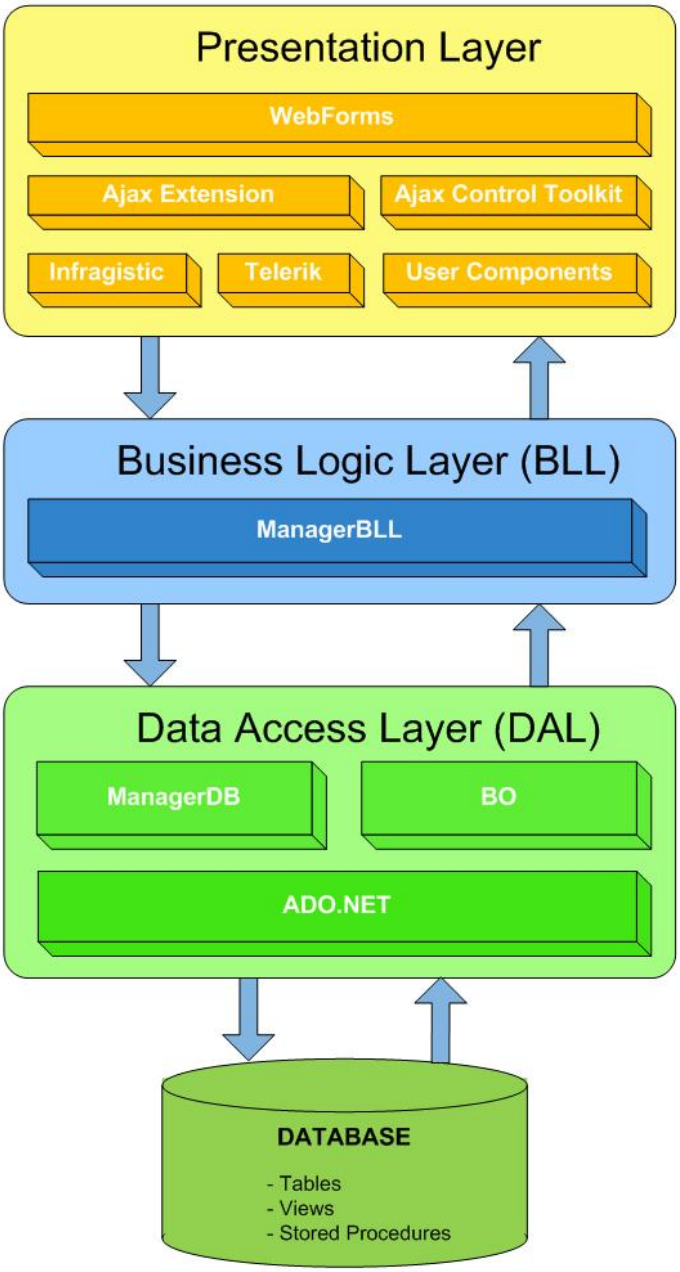
5.2.1 Prezentační vrstva

Už z názvu vrstvy vyplývá, že půjde o uživatelské rozhraní aplikace a použité technologie v ní. Jádro prezentační vrstvy tvoří ASP.NET technologie, která určuje základní zobrazené ovládací prvky. Doplňkem pro zajištění asynchronní komunikace jsou využité AJAX knihovny, které se navazují a spolupracují s ovládacími prvky ASP.NET.

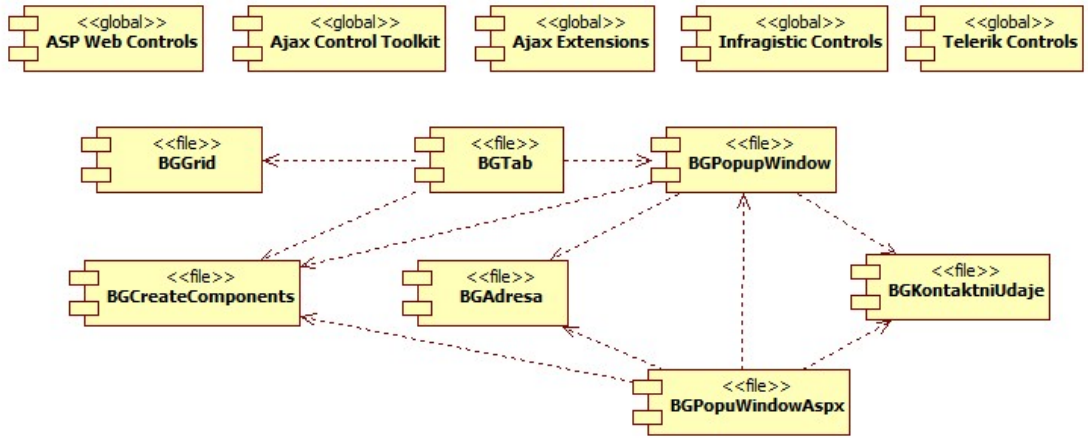
Pro rozsáhlejší a efektivnější využití těchto dvou technologií jsou do návrhu přidány licencované komponenty firem Infragistics a Telerik. Obě komponenty poskytují podobné interaktivní ovládací prvky, které jsou často poptávány u webových aplikací. Jejich zahrnutí slouží pro pozdější sledování jejich použití a spolupráci s ostatními komponentami. Později výsledek sledování firmě ukáže, na kterou komponentu uplatnit licenci.

Poslední uživatelské komponenty seskupují všechny zmíněné technologie do jednoho použitelného rámce. Skládají se z podpůrných komponent, které výslednou komunikací mezi sebou tvoří požadovaný vzhled záložky či formuláře systému.

Diagram komponent na obrázku 14 ukazuje komunikaci použitých komponenty určujících vzhled a funkce prezentační vrstvy. Lze na něm vidět, jak uživatelsky vytvořené komponenty využívají rysy každé technologické komponenty mající v diagramu stereotyp `<<global>>`.



Obrázek 13: Vrstvená architektura systému BIGGIE



Obrázek 14: Diagram komponent prezentační vrstvy

5.2.2 Byznys vrstva (BLL)

Byznys vrstva odpovídá za přístup do vrstvy logiky řízení dat, a za zpracování údajů přijatých a odeslaných do prezentační vrstvy. Hlavní úkol vrstvy spočívá ve validaci a provádění byznys procesů. Každý podstatný objekt v systému má svého manažera, který funguje jako transparentní vrstva mezi prezentační vrstvou a vrstvou pro přístup k datům, a komunikuje s ostatními manažery k zajištění provedení byznys procesu.

5.2.3 Vrstva logiky řízení dat (DAL)

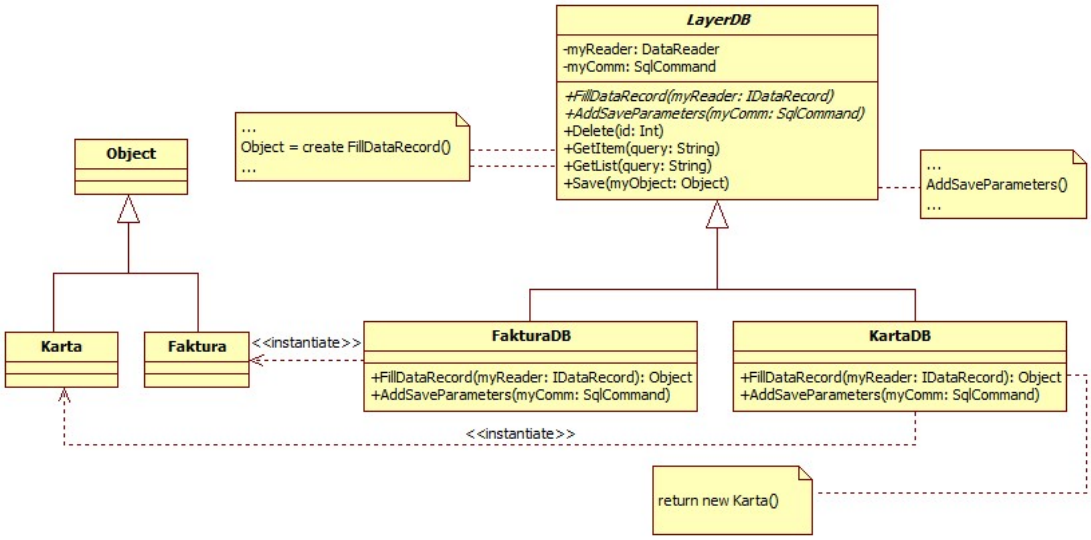
V návrhu přístupu k databázi byly identifikovány metody pro manipulaci s daty společné pro všechny byznys objekty. Jedná se o vkládání, editaci, mazání a výpisy záznamů. Vrstvu charakterizuje přístup k datům přes ADO.NET technoligii, která k tomu využívá třídy `DataReader`. Podporu připojení k zadanému relačnímu databázovému systému SQL Server poskytuje knihovna `SqlClient`.

Nad třídním diagramem popisujícím vrstvu logiky přístupu k datům došlo k využití návrhových vzorů strukturálních - Tovární metoda a chování - Šablonová metoda. Bylo nutné vytvořit model pro efektivní přístup k datům a zajistit jejich mapování na příslušné byznys objekty. Popis principu použitých návrhových vzorů obsahují další odstavce.

Užité návrhové vzory

Účelem návrhového vzoru Tovární metoda je definovat rozhraní pro vytvoření objektu, přičemž o to, kterou instacující třídu se jedná, rozhoduje odpovídající podtřída [6]. Použití vzoru se nabídlo při procesu plnění byznys objektů načítaných z databáze, kde poté došlo k jejich předání byznys vrstvě.

Návrhový vzor Šablonová metoda definuje kostru algoritmu tím, že některé kroky nechává na potomcích. Umožňuje tak potomkům upravit určité kroky algoritmu bez toho, aby měnili strukturu algoritmu. Využití toho návrhového vzoru poskytlo rámec pro vkládání a editaci záznamů do databáze.



Obrázek 15: Třídní diagram datové vrstvy

Tovární metoda se stará o plnění atributů byznys objektů prostřednictvím abstraktní metody `FillDataRecord()` užívanou metodami `GetItem()` a `GetList()`. Potomek implementuje plnění objektu svého typu příchozím objektem `DataReader` a vrací jej zpět do metod, které pracují s obecným typem `Object`. Šablonová metoda se využívá u procesu ukládání záznamů (metoda `Save()`). Přesněji každý potomek přepisuje metodu `AddSaveParameters()`, která do objektu typu `SqlCommand` přiřadí parametry, jenž požaduje uložená procedura na databázovém serveru. Diagram popisuje obrázek 15.

5.2.4 Byznys objekt (BO)

Byznys objekt reprezentuje třída poskytující soukromé proměnné, ke kterým lze přistoupit přes viditelné přístupové metody. Vrstvou logiky přístupu k datům mapuje hodnoty atributů databázové tabulky na soukromé proměnné byznys objektu. Soukromé proměnné mohou být libovolně přidávány a nemusí být omezeny počtem atributů databázové tabulky.

5.3 Návrh implementace logiky řízení dat (DAL)

Vrstva logiky přístupu k datům (DAL) obsahuje hlavní třídu `LayerDB`, z které dědí ostatní databázoví manažeři. Potomci implementují svůj specifický kód do abstraktních metod `FillDataRecord()` a `AddSaveParameters()`, čímž získají funkční klíčové metody pro přístup k databázi - `GetItem()`, `GetList()`, `Save()` a `Delete()`. Popis třídy ukazuje diagram na obrázku 15 výše.

Pro přístup k databázi se využívá souboru knihoven ADO.NET. Nedochází k plnění `DataSetu` prostřednictvím objektu `SqlDataAdapter`, neb se využívá byznys objektů. Došlo k využití tří níže popsaných tříd knihovny `System.Data.SqlClient`:

- `SqlConnection` - reprezentuje objekt pro připojení k SQL Server databázi.
- `SqlCommand` - reprezentuje SQL příkaz nebo uloženou proceduru prováděnou nad SQL Server databázi, kde pro jejich spuštění se využívá metod `ExecuteReader()` pro čtení a `ExecuteNonQuery()` pro zápis.
- `SqlDataReader` - poskytuje způsob jak jednosměrně číst řádky z výsledků dotazů nad SQL Server databází.

5.3.1 Konfigurace připojení

Třída `AppConfiguaration` obsahuje vlastnosti určené pro čtení částí nastavení z konfiguračního souboru aplikace `web.config`. Přístup do něj nabízí statická třída ASP.NET `ConfigurationManager` pod proměnnou `ConnectionStrings` zpřístupňuje kolekci definovaných připojení. Využitím třídy `AppConfiguaration` v databázových manažerech získáváme nezávislou část aplikace pro změnu připojení k různým SQL Server databázím (ukazuje výpis kódu 2).

```
public static class AppConfiguration
{
    ...

    public static string ConnectionString
    {
        get
        {
            return ConfigurationManager.ConnectionStrings[”ERPLIVE_XEVOS”].ConnectionString;
        }
    }
}
```

Výpis 2: Ukázka kódu třídy AppConfiguration

5.3.2 Komunikace DAL s ADO.NET třídou LayerDB

Databázoví manažeři neboli třídy pro řízení přístupu k datům, dědí základní operace z třídy `LayerDB`. Abstraktní třída `LayerDB` slouží pro implementaci společných metod všech objektů nad operacemi databáze, které budou probrány níže. `FillDataRecord()` metoda slouží pro naplnění byznys objektu z příchozího záznamu typu `DataRecord`. V procesu ukládání se využívá metoda `AddSaveParameters()`, jenž naopak příchozí parametr `comm` naplní daty z konkrétního byznys objektu podle třídy, která tuto metodu implementuje.

```
public List<Object> GetList(string query)
{
    List<Object> tempList = null;
    using (myConnection = new SqlConnection(AppConfiguration.ConnectionString))
    {
        SqlCommand myCommand = new SqlCommand(query, myConnection);

        try {
            myConnection.Open();
            using (SqlDataReader myReader = myCommand.ExecuteReader())
            {
                tempList = new List<Object>();
                if (myReader.HasRows)
                {
                    while (myReader.Read())
                    {
                        tempList.Add(FillDataRecord(myReader));
                    }
                }
            }
        }
        catch (SqlException err) {
            // Nahrazeni chybove zpravy necim vice konkretnim.
            // Zde moznost chybu zaprotokolovat.
            throw new ApplicationException("Data error." + err.Message);
        }
        finally {
            myConnection.Close();
        }
    }

    return tempList;
}
```

Výpis 3: Výpis seznamu objektů ve třídě LayerDB

Metoda `GetList()` provede inicializaci readeru `SqlDataReader`, a dokud obsahuje řádek, tak jej posílá do již konkrétní implementace metody `FillDataRecord()`. Zdali reader nenačetl řádek z výsledku dotazu, určuje metoda `Read()` tím, že vrací `false`. Nakonec metoda vrátí seznam objektů pro přetypování na konkrétní typ, což ukazuje výpis kódu 3.

```
public int Save(Object myObject, int objectId, string tableName)
{
    int result = 0;
    // inicializace pripojeni
    myConnection = new SqlConnection(AppConfiguration.ConnectionString);

    StringBuilder spName = new StringBuilder();
    spName.Append("sp");
    spName.Append(tableName);

    SqlCommand myCommand;
    if (objectId == 0 || objectId == -1)
    {
        spName.Append("Insert");
        myCommand = new SqlCommand(spName.ToString(), myConnection);
        myCommand.Parameters.Add("@id", SqlDbType.Int);
    }
    else
    {
        spName.Append("Update");
        myCommand = new SqlCommand(spName.ToString(), myConnection);
        myCommand.Parameters.AddWithValue("@id", objectId);
    }
    myCommand.CommandType = CommandType.StoredProcedure;
    myCommand.Parameters["@id"].Direction = ParameterDirection.InputOutput;

    // Abstraktni metoda vraci SqlCommand doplneny potomkem o specificke parametry
    myCommand = AddSaveParameters(myObject, myCommand);

    try {
        myConnection.Open();
        myCommand.ExecuteNonQuery();
        result = (int)myCommand.Parameters["@id"].Value;
        myConnection.Close();
    }
    ...
    return result;
}
```

Výpis 4: Uložení a editace ve třídě LayerDB

Výpis kódu 4 ukazuje metodu `Save()` abstraktní třídy `LayerDB`, jenž spojuje kód pro operace uložení a aktualizování záznamu, kde se pro uložení rozhodne v případě nezáporné hodnoty v identifikačním čísle ukládaného objektu (`objectID`). Uložení a aktualizace záznamu spadá pod využití uložených procedur, které splňují výše zmíněnou šablonu jejich názvu `sp[Nazev_tabulky][Nazev_operace]`, a lze tedy programově takový název složit s využitím parametru `tableName`. Po naplnění parametrů potomkem do proměnné typu `SqlCommand` metodou `AddSaveParameters()`, se provede vykonání uložené procedury metodou `ExecuteNonQuery()` a vrátí se počet ovlivněných záznamů.

```
public int DeleteViaUpdate(string modulName, List<int> myListId)
{
    int result = 0;
    string query = String.Empty;
    SqlCommand myCommand;

    if (myListId.Count == 1)
        query = "UPDATE " + modulName + @" SET smazat = 'True', zmeneno = @zmeneno
            WHERE id =" + myListId[0].ToString();
    else
        query = "UPDATE " + modulName + @" SET smazat = 'True', zmeneno = @zmeneno
            WHERE (id IN (" + PomocneFunkce.getSeparateStrings(myListId) + "))";

    myCommand = new SqlCommand(query, myConnection);
    // Pridat navic zmeneno do zaznamu, ktere jsou nastaveny na smazani
    myCommand.Parameters.AddWithValue("@zmeneno", DateTime.Now);

    try {
        myConnection.Open();
        result = myCommand.ExecuteNonQuery();
    }
    catch (SqlException err) {
        // Nahrazeni chybove zpravy necim konkretnim. Zde chybu zaprotokolovat.
        throw new ApplicationException("Data error." + err.Message);
    }
    finally {
        myConnection.Close();
    }
    // vraceni vysledku ExecuteNonQuery()
    return result;
}
```

Výpis 5: Mazání ve třídě LayerDB

Výpis kódu 5 představuje metodu `DeleteViaUpdate()` abstraktní třídy `LayerDB`, jenž neprovádí vymazání záznamu nalezeného podle identifikačního čísla, ale vykoná jeho aktualizování na neplatný a zaeviduje časovou změnu. Parametr `myListId` určuje, že lze volat mazání více záznamů. Mazání se vykoná jedním dotazem s využitím metody `getSeparateStrings()`, která seznamu identifikačních čísel vrátí oddělené čárkou v podobě textu.

5.3.3 Konkrétní manažer pro DAL

Výpis kódu 6 ukazuje implementaci abstraktních metod. Metoda `FillDataRecord()` vytvoří instanci konkrétního byznys objektu pro namapování jeho atributů sloupci z `IDataRecord`. Při načítání se musí hlídat, zdali nejsou vrácená data prázdná (`null`) přes metodu `IsDBNull()`. Byznys objekt se vrátí jako typ `Object`, aby s ním mohla pracovat volající metoda třídy `LayerDB`.

```
namespace Biggie.Zasoba.DAL
{
using Biggie.Zasoba.BO;
using System.Data.SqlClient;

// Databazovy manager KartaDB, dedi zakladni metody z abstraktni tridy LayerDB
public class KartaDB : LayerDB
{
    ...
    // implementace metod k prepsani od LayerDB
    protected override Object FillDataRecord(IDataRecord myDataRecord)
    {
        Karta myKarta = new Karta(); // novy objekt k naplneni
        ...
        myKarta.Id = myDataRecord.GetInt32(myDataRecord.GetOrdinal("id"));
        // zjisteni DBNull pro String, pokud ano ponecha se inicializacni hodnota String.Empty
        if (!myDataRecord.IsDBNull(myDataRecord.GetOrdinal("popis")))
        {
            myKarta.Popis = myDataRecord.GetString(myDataRecord.GetOrdinal("popis"));
        }
        ...
        return myKarta;
    }

    protected override SqlCommand AddSaveParameters(Object obj, SqlCommand comm)
    {
        Karta myKarta = (Karta)obj; // pretypovani na konkretni typ
        ...
        comm.Parameters.AddWithValue("@nazev", myKarta.Nazev_karta);// plneni parametru
        if (myKarta.Id_vyrobku == -1)
            comm.Parameters.AddWithValue("@id_vyrobku", DBNull.Value);
        else
            comm.Parameters.AddWithValue("@id_vyrobku", myKarta.Id_vyrobku);
        ...
        return comm;
    }
}
}
```

Výpis 6: Implementace abstraktních metod potomkem KartaDB

Metoda `AddSaveParameters()` přetypuje poslaný objekt k uložení na konkrétní typ byznys objektu. Provede plnění parametrů objektu `SqlCommand`, jenž je vrácen zpět do volající metody třídy `LayerDB`. Při plnění parametrů pro uložení dochází k řízení nepovinných atributů databázové tabulky, kterým se v případě prázdné nebo výchozí hodnoty nastaví `DBNull.Value`.

6 Implementace

Cílem kapitoly není podrobný popis využitých technologií, ale pouze ukázat vytvořené programové části nad systémem BIGGIE podle zvoleného návrhu. Vysvětlení tedy bude směrováno na důležité implementace v systému, ve kterých se počítá se znalostí základů ASP.NET.

6.1 Aplikační logika (BLL)

Část aplikační logika popisuje použité praktiky k předání žádaných byznys objektů nebo informací prezentační vrstvě a navazuje tak na vrstvenou architekturu v návrhu systému. Soubory aplikační logiky a logiky pro řízení dat jsou v ASP.NET zařazeny do složky `App_Code`.

6.1.1 Knihovny ComponentModel pro výpisy manažera BLL

Prezentační vrstva si může žádat o data prostředníctím jejího ovladače, který zajistí naplnění příslušných ovládacích prvků pro zobrazení dat. Ovladač může tuto funkci provádět ručně (uživatelův kód) nebo automaticky odkazem na zdroj dat. Jak již bylo zmíněno, použitým zdrojem dat v prezentační vrstvě je `ObjectDataSource`, kterému lze určit metody pro práci s daty přes manažera vrstvy aplikační.

K zajištění toho, aby měl ovládací prvek `ObjectDataSource` k dispozici třídu manažera vrstvy BLL, se využívá .NET framework knihovny `System.ComponentModel`, která poskytuje třídy užívané k implmentaci chování komponent nebo oládacích prvků. Třídu uvozenou atributem `DataObjectAttribute()` identifikuje `ObjectDataSource` jako možný objekt k jeho naplnění.

Dále je nutné identifikovat metody této třídy, které slouží pro obsluhu metod ovládacího prvku `ObjectDataSource`. Mezi metody patří smazání, vložení, výpis jednoho objektu a výpis několika objektů. Knihovna `ComponentModel` obsahuje třídu pro mapování metod zdroje dat - `DataObjectMethodAttribute`. Tato třída identifikuje metody podle prováděné datové operace atributem `DataObjectMethodType`. Navíc lze v konstruktoru nastavit výchozí metodu pro danou operaci, v případě jejich přetížení.

Použití modelu prezentuje výpis kódu 7, který navíc ukazuje komunikaci s vrstvou logiky řízení dat. Vrstva logiky řízení dat vrací objekt, který je nutné přetypovat na konkrétní typ a vrátit prezentační vrstvě.

Třída `KartaManager` představuje aplikační logiku objektu `Karta` a je umístěna ve jmenném prostoru `Biggie.Zasoba.BLL`. Ke komunikaci s vrstvou logiky řízení dat je nutné připojit jmenný prostor `Biggie.Zasoba.DAL`. Pro předání a práci nad byznys objeketem se připojuje jmenný prostor `Biggie.Zasoba.BO`. Takové logické členění do jmenných prostorů se využívá pro každý byznys objekt.

```
namespace Biggie.Zasoba.BLL
{
    using Biggie.Zasoba.BO;
    using Biggie.Zasoba.DAL;
    using System.ComponentModel;

    [DataObjectAttribute()]
    public class KartaManager
    {
        // stejny s nazvem tabulky v databazi
        private static readonly String tableName = "Karta";
        // promenna jejiz hodnota uchovava nazev modulu
        private static readonly String modulName = "ViewKarta";

        //databazovy manager
        private KartaDB kartaDB;

        // Vraci jeden objekt typu Karta podle ID.
        [DataObjectMethod(DataObjectMethodType.Select, true)]
        public Karta GetItem(int id)
        {
            Karta karta = null;    // nacteni karty
            if (id != 0) karta = (Karta)kartaDB.GetItem(id, tableName, queryDetail);
            return karta;
        }

        // Vraci seznam objektu typu Karta.
        [DataObjectMethod(DataObjectMethodType.Select, true)]
        public List<Karta> GetList()
        {
            // deklarovani promenne pred jejim pouzitim
            string queryPrint = "SELECT * FROM " + modulName;
            // nunte seznam objektu pretypovat metodou ConvertAll na seznam karet
            List<Karta> kartaList = kartaDB.GetList(queryPrint).ConvertAll<Karta>(delegate(
                object o) { return (Karta)o; });
            return kartaList;
        }
    }
}
```

Výpis 7: Vzor manažera na třídě KartaManager s metodami pro výpis

6.1.2 Zpracování uložení a mazání manažerem BLL

Byznyz vrstva provádí logiku pro zpracování požadavku prezentační vrstvy s využitím dotazování se na data vrstvy logiky řízení dat. Výpis kódu 8 ukazuje řízení uložení objektu `Karta`, kde, kromě konečného uložení karty, dochází k uložení objektů na kartu vázaných s využitím komunikace s příslušnými datovými vrstvami. Metoda `Save()` i `Update()` volají stejnou metodu vrstvy logiky řízení dat, neboť až v ní se rozhodne typ operace podle hodnoty identifikačního čísla.

```
[DataObjectMethod(DataObjectMethodType.Insert, true)]
public int Save(Karta myKarta)
{
    /* DOBA ZARUKY - kontrola zda byl podobjekt nacten z existujiciho zaznamu(ma ID > 0)*/
    if (myKarta.Id_zaruky < 1)
    {
        DobaZarukyDB dobaZarukyDB = new DobaZarukyDB();
        myKarta.Id_zaruky = dobaZarukyDB.Save(myKarta.DobaZarukyBO, myKarta.Id_zaruky, "
            Doba_zaruky");
    }
    /* IDENTIFIKACE */
    if (myKarta.Id_identifikace < 1)
    {
        IdentifikaceZboziDB identifikaceDB = new IdentifikaceZboziDB();
        myKarta.Id_identifikace = identifikaceDB.Save(myKarta.IdentifikaceZboziBO, myKarta.
            Id_identifikace, "Identifikace_zbozi");
    }

    myKarta.Id = kartaDB.Save(myKarta, myKarta.Id, "Karta");
    // navratova hodnota metody save(), takze ID nove karty
    return myKarta.Id;
}

[DataObjectMethod(DataObjectMethodType.Update, true)]
public int Update(Karta myKarta, int idUser)
{
    /* stejny proces jako Save metoda, kde se nakonec vola Save metoda */
    // V metode Save se podle hodnoty ID rozhodne zda jde o vlozeni ci aktualizaci
    myKarta.Id = kartaDB.Save(myKarta, myKarta.Id, "Karta");
    // navratova hodnota ID
    return myKarta.Id;
}
```

Výpis 8: Uložení ve třídě KartaManager

Výpis kódu 9 ukazuje provádění logiky pro mazání záznamu s kontrolou, kterou realizuje metoda `CanDelete()`. Navíc poskytuje dvě možnosti smazání záznamu, kde metodou `Delete()` s parametrem vybraných karet provede databázová vrstva pouhé zneplatnění záznamu. Metoda `DeleteWithoutRecovery()` může být řízena zdrojem dat, proto se označí atributem typu metody jako `DataObjectMethodType.Delete` a zároveň se nastaví jako metoda výchozí - `true`.

```
public int Delete(List<int> myKartaList)
{
    if (CanDelete(myKartaList) > 0)
        return -1;

    /* DELETE je realizovano pro uzivatele zneplatnenim zaznamu, tudiz UPDATE */
    return kartaDB.DeleteViaUpdate(tableName, myKartaList);
}
```

```
[DataObjectMethod(DataObjectMethodType.Delete, true)]
public bool DeleteWithoutRecovery(Karta myKarta)
{
    if (CanDelete(myKartaList) > 0)
        return -1;
    /* DELETE je realizovan ostrym smazanim zaznamu*/
    return kartaDB.Delete(myKarta.Id, tableName);
}
```

Výpis 9: Mazání ve třídě KartaManager

6.1.3 Transakce při uložení a mazání

```
using (System.Transactions.TransactionScope ts = new System.Transactions.TransactionScope()){
    try {
        List<Faktura> fakturyKeSmazani = GetItemsByListID(myFakturaSeznam, typ_dokladu ?
            BGNazvy.OBJECT_NAME_FAKTURA_PRIJATA : BGNazvy.
            OBJECT_NAME_FAKTURA_VYDANA); // nutne nacist si objekty faktury

        // 1. smazani pres aktualizaci vsech faktur
        int results = fakturaDB.DeleteViaUpdate(BGNazvy.TABLE_NAME_FAKTURA,
            myFakturaSeznam);

        PolozkaNpoDB polozkaNpoDB = new PolozkaNpoDB();// DB manager pro polozky

        foreach (Faktura fakturaSmazani in fakturyKeSmazani)
        {
            List<PolozkaNpo> polozkyFaktury = new FakturaDetailManager().
                GetListPolozkyFaktura(fakturaSmazani.Id); // nacteni polozek
            if (polozkyFaktury != null)
            {
                // nastaveni polozek na zneplatnene
                foreach (PolozkaNpo polozka in polozkyFaktury) {
                    polozka.Smazat = true;
                }

                // 2. metoda provede standardni zneplatneni polozek
                polozkaNpoDB.SmazZmenExistujiciPolozkyDokladu(polozkyFaktury, BGNazvy.
                    OBJECT_NAME_FAKTURY, String.Empty, fakturaSmazani.Id, id_zapisovatele,
                    DateTime.Now, fakturaSmazani.Typ_faktury);
            }
        }
        ts.Complete(); // uspesne ukonceni transakce
    }
    catch (Exception ex) {
        // provede se automaticky rollback, pokud nebyl v bloku transScope vyvolana Complete()
        return BGNazvy.hlaska_NedosloKeSmazaniDokladu;
    }
}
```

Výpis 10: Ukázka transakčního zpracování procesu mazání faktur

Uložení (mazání) jednoho záznamu odpovídá transakční vykonání dotazu nad SQL Serverem. Z používání komplexnějších a pro systém podstatných byznys objektů vyplynula nutnost zajištění uložení (mazání) všech byznys objektů týkajících se prováděného procesu - především dokladů včetně kolekce jejich položek. Tento požadavek vedl k využití transakcí.

Už verze .NET frameworku 2.0 poskytuje jmenný prostor `System.Transactions`, jenž byl využit z důvodu zachování struktury akutálních aplikačních kódů. Alternativu představuje transakce na straně SQL Serveru, kde by se dotazy, pod ní patřící, prováděly v uložené proceduře, což by vedlo k nutnosti přizpůsobení aplikační logiky. Ze stejného důvodu se nevyužila třída `SqlTransaction` prostoru `System.Data.SqlClient`, jejíž instance se připojuje do prováděných příkazů typu `SqlCommand`.

V ukázce kódu 10 sledujeme transakční zpracování mazání seznamu faktur včetně jejich seznamu položek, kde se záznamy zneplatňují. U seznamu položek se volá speciální metoda pro ovlivnění přepočtů či množství na skladě. Využitím třídy `TransactionsScope` z jmenného prostoru `System.Transactions` došlo k určení bloku, který má být transakčně prováděn. Takovou abstrakci si lze dovolit na jakékoliv úrovni kódu, kde v našem případě došlo k využití v byznys logice systému, přesněji u manažerů dokladů.

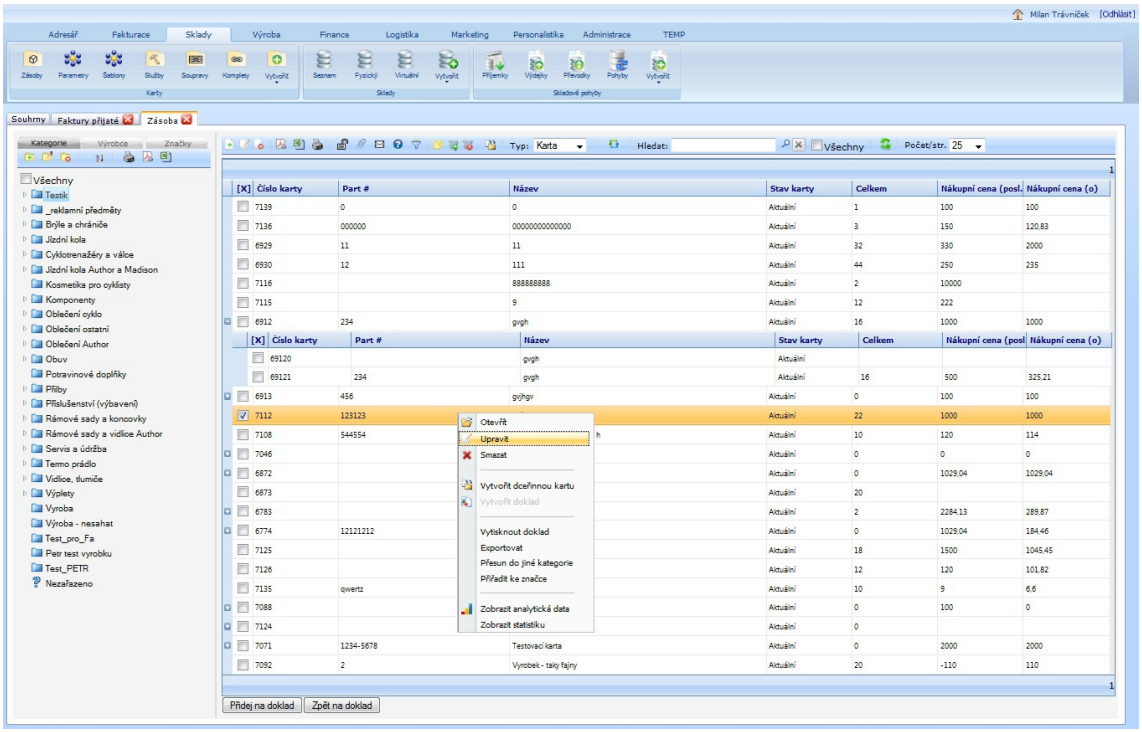
Součástí transakčního bloku jsou příkazy `try-catch` zajišťující odchycení chyby a následného nedokončení transakce metodou `Complete()`, která musí být volána na konci procesu. Dojde-li k zachycení vzniklé chyby s opuštěním bloku `TransactionsScope` bez potvrzení ukončení, automaticky se provede zpětné volání nad dotazy (rollback).

Poznámka 6.1 Pro bezchybovou funkčnost užitých transakcí je nutné na serveru s aplikační logikou nastavit automatické spouštění služby Koordinátor distribuovaných transakcí.

6.2 Vzhled systému

Implementace prezentační vrstvy zakládá především na dynamickém kódu, který řídí třídy uložené v `App_Code`. Mými ostatními spolupracovníky z firmy XEVOS Solutions s.r.o. byly navrženy šablonové třídy obsahující abstrakce kódů a realizující požadovaný návrh systému. Třídy byly průběžně podle potřeb doplňovány a upravovány všemi účastníky projektu, patří mezi ně:

- `BGTab` - obecná třída obsahující společné kódy pro všechny záložky systému, kde specifické vlastnosti či vzhled si určí potomek sám. Vrací připravenou ASP.NET záložku typu `TabPanel`, která se přidává do kontejneru `TabContainer` umístěným na stránce `Default.aspx`.
- `BGPopupWindow` - definuje obecnou třídu pro vytvoření modálního okna, které vychází z využití `ModalPopupExtenderu` knihovny AJAX Control Toolkit.



Obrázek 16: Ukázka vzhledu systému

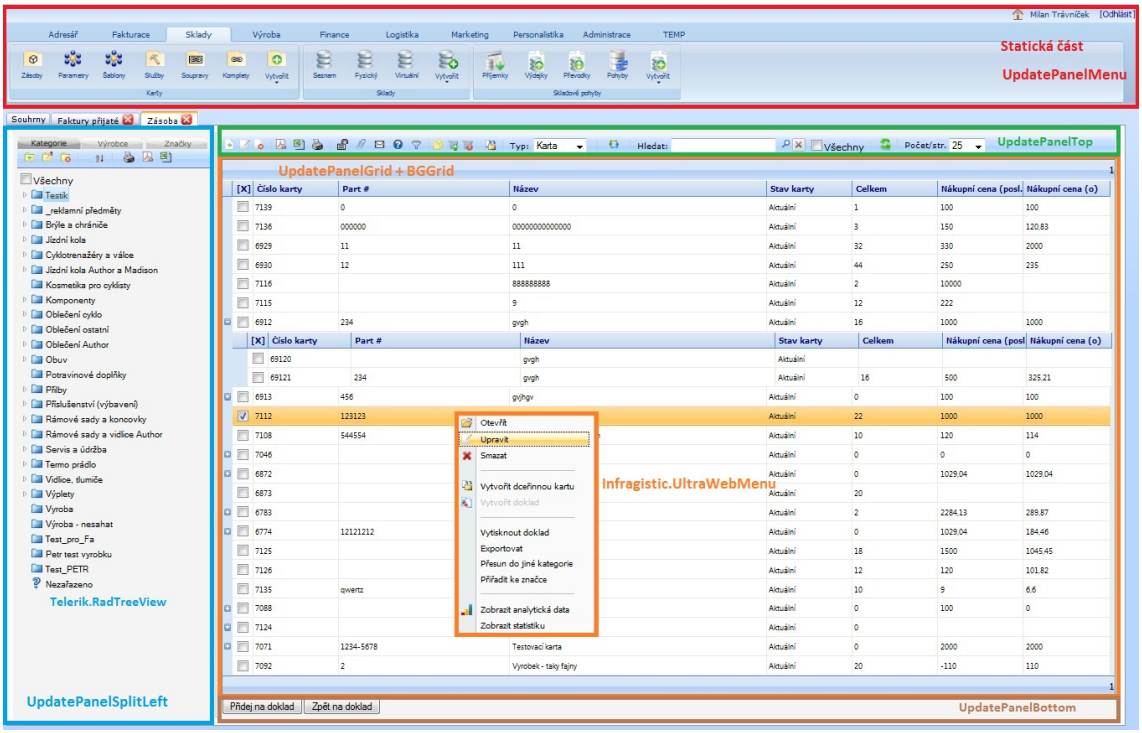
- `BGPopupWindowASPX` - definuje obecnou třídu pro vytvoření samostatného okna pomocí ovládacího prvku `Panel`, který se zobrazí v novém okně prohlížeče vyvoláním JavaScriptového kódu.

Veškeré řízení záložek systému spadá pod stránku `Default.aspx`. Jako statické kódy vzhledu, umístěných na stránce `Default.aspx`, byly identifikovány následující neměnné části systému - horní menu a kontejner pro záložky, kde oba dva prvky patří pod vlastní `UpdatePanel`. Návrh a implementace menu nespadaly do mého účelu práce a pouze jsem na něj navázal funkčnost vyvolání záložky.

Mým úkolem ve firmě, v rámci práce nad prezentační vrstvou, bylo využití výše zmíněných podpůrných knihoven a daných principů k napojení této vrstvy na mnou vybudovanou aplikační logiku. Dále tedy zmiňuji pouze části kódu, na kterých jsem se minimálně větší měrou podílel.

6.2.1 Proces vytvoření záložky

Obrázek 17 ukazuje rozdělení stránky `Default.aspx` jednotlivými UpdatePanely, které odpovídají za svou funkčnost. Vše se vytváří dynamickým kódem (mimo statickou část), jehož řízení se rozděluje mezi `Default.aspx.cs` a konkrétní instance šablonové třídy `BGTab`, jejíž funkčnost lze stručně popsat pouze obecným postupem:



Obrázek 17: Schéma rozdělení záložky

1. `BGTab` poskytuje metodu `CreateTabWithSplitter()` (vytvoření záložky s levým panelem) a `CreateTabWithoutSplitter()` (bez levého panelu), kde obě metody vykonávají přidání UpdatePanelů s příslušnými ovládacími prvky pro horní operace, grid a dolní operace (viz obrázek 16).
2. `BGTab` dále poskytuje potomkům metodu `GenerateGrid()`, která provede inicializaci gridu a přidá jej na `UpdatePanelGrid`.
3. Potomek třídy `BGTab`, konkrétně `BGTabZasoba` (podle obrázku 16), inicializuje své specifické ovládací prvky pro horní a dolní část panelu.
4. `BGTabZasoba` už jen zavolá připravené metody třídy `BGTab` s danými prvky určující jeho typ. Pro záložku zásoby jde o metodu `CreateTabWithSplitter()` a metodu pro vygenerování gridu - `GenerateGrid()`.
5. Nakonec potomek vrátí naplněný objekt typu `TabPanel` do prezentační vrstvy, která si jej vyžádala.

```
public void GenerateGrid(string modulName, string viewName, string objectName, string
     objectNamespace)
{
    ...
    UpdatePanelGrid.ContentTemplateContainer.Controls.Clear(); // nunte vymazani obsahu UP

    switch (modulName) // nalezeni typu gridu podle nazvu modulu
    {
        ...
        case "Zasoba":
            // inicializace
            grid = new BGGrid(objectName, objectNamespace, Key, GetODSParameters(
                parameters));

            UpdatePanelGrid.ContentTemplateContainer.Controls.Add(grid.
                objectDataSourceMaterske);
            UpdatePanelGrid.ContentTemplateContainer.Controls.Add(grid.
                objectDataSourceDcerinne);
            UpdatePanelGrid.ContentTemplateContainer.Controls.Add(grid.hierarchicalDataSource)
                ;
            break;
        ...
    }
    // pridani gridu typu Infragistic na UPGrid
    UpdatePanelGrid.ContentTemplateContainer.Controls.Add(grid.ultraWebGrid);
    ...
    // plneni datoveho zdroje
    grid.hierarchicalDataSource.DataBind();
    grid.ultraWebGrid.Update();

}
```

Výpis 11: Ukázka procesu generování gridu

Ukázka kódu 11 zachycuje důležité kroky při inicializaci gridu pro záložku. Třída `BGGrid` poskytne inicializaci ovládacích prvků typu zdroje dat (ObjectDataSource) a grid pro přidání na `UpdatePanelGrid`. Inicializaci poskytne třída `BGCreateComponents`. Finální aktualizaci UpdatePanelu, po `DataBind()` metodě, se předají objekty z aplikační logiky do zdroje dat, který jimi naplní grid.

6.2.2 Proces vyvolání záložky

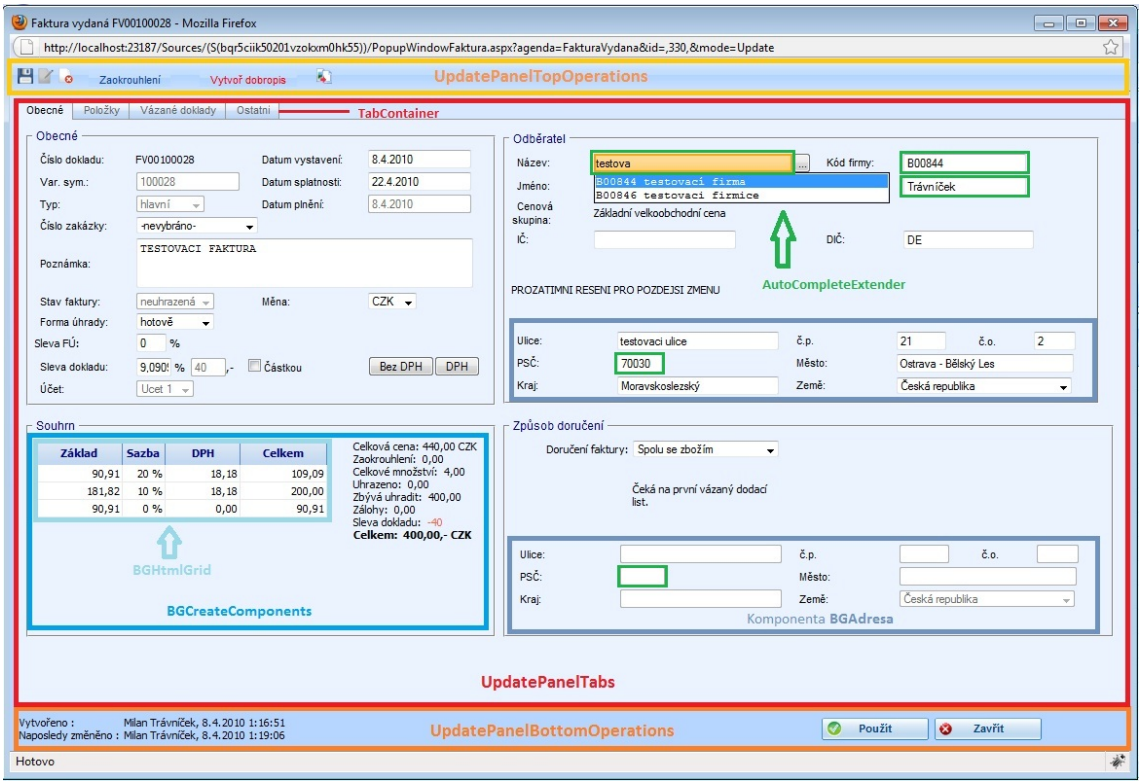
```
protected void load_my_subpage(object sender, EventArgs e)
{
    switch (((Button)sender).ID.ToString()) {
        ...
        case "karta_zasoba":
            AddTabZasoba(null); // vytvori a prida TabZasobu do TabContaineru
            break;
        ...
    }
    ...
    UpdatePanelZalozky.Update();
}

private void AddTabZasoba(BGTabZasoba tab)
{
    // urceni poradi podle toho zda existuje
    ...
    bgTabZasoba = new BGTabZasoba("tab" + poradi, BGNazvy.TAB_ZASOBA_TITLE, BGNazvy.
        OBJECT_NAME_ZASOBA);
    // registrace udalosti
    bgTabZasoba.imgCloseButton.Click += new ImageClickEventHandler(imgCloseButton_Click);
    ...
    // inicializace tabPanelu
    tabPanel = bgTabZasoba.CreateTabZasoba();
    ...
    // pridani do TabKontaineru
    ZalozkyTab.Tabs.Add(tabPanel);
    if (active) // nastaveni aktivni zalozky
        ZalozkyTab.ActiveTab = tabPanel;
}
```

Výpis 12: Proces vygenerování záložky řízený z Default.aspx.cs

Výpis kódu 12 začíná odchycením kliknutí na tlačítko záložky zásob ze statického menu. Metoda `load_my_subpage()` podle identifikátoru rozhodne, která záložka se má vyvolat. Po vytvoření a přidání záložky do `TabContainer` metodou `AddTabZasoba()`, se musí zavolat aktualizace UpdatePanelu záložek.

S dynamickým kódem přichází zodpovědnost za řízení záložek či všech dynamicky vytvořených objektů. V případě dalšího zpracování požadavku, se nejdříve vyvolá PostBack, který provede vykreslení stránky `Default.aspx` v její statické podobě na serveru.



Obrázek 18: Záložka Obecné formuláře faktury

Tím se ztratí vyvolaná záložka z důvodu prázdného kontejneru (výchozí nastavení při přidání na stránku).

Řešení poskytuje přepsaná metoda `OnInit()`, jež za pomoci příznaku o existenci záložky (uloženo v paměti Cache), volá metodu `CreateExistsTabs()`. `CreateExistsTabs()` nedělá nic jiného než nalezení uživatelem otevřených záložek v paměti a provedení jejich inicializačních metod, v našem případě metoda `AddTabZasoba()`.

6.2.3 Detail faktury v samostatném okně

Pro ukázku implementace detailu záznamu se nabízí samostatné okno faktury na obrázku 18. Schéma ukazuje rozložení UpdatePanelů na okně, využití podpůrných komponent a sestavení vzhledu. Komponenta `BGCreateComponents` využívá další komponentu `BGHtmlGrid` k sestrojení souhrnů, na kterých je vidět i správná funkčnost uplatnění slev. Horní UpdatePanel obsahuje zleva tlačítka pro - uložení dokladu, povolení editace dokladu, vyvolání modálního okna pro zaokrouhlení či funkce pro převedení dokladu na dodací list.

6.2.3.1 AutoCompleteExtender Inicializace prvku `AutoCompleteExtender` z knihovny AJAX Control Toolkit se provede ve třídě `BGPopupWindowFaktura` v metodě `Render()`, což ukazuje kód 13. Své využití nachází při svázaní s prvkem `TextBox` (důležité proměnné - `ServiceMethod`, `ServicePath`, `TargetControlID`)), který poté využívá webovou službu pro získání dat.

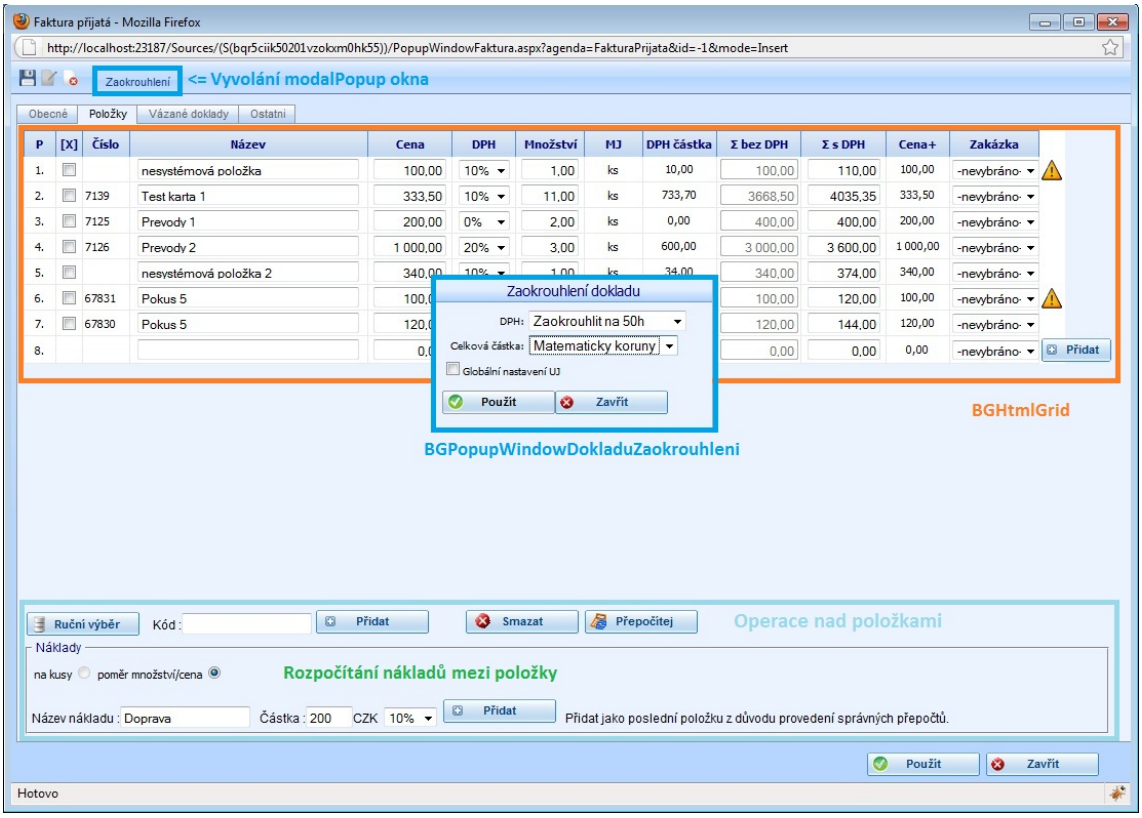
Webová služba byla nastavena jako statická metoda s povolením užívání Session a vložena do kódu na pozadí stránky faktury (`PopupWindowFaktura.aspx.cs`). Naznačení jejích kroků ukazuje kód 14. `AutoCompleteExtender` se využívá také v podpůrné komponentě `BGAdresa`, kde již obsahuje vlastní webové služby pro plnění dat podle psč kódu.

```
public void Render()
{
    // BGCreateComponents obsahuje dynamicky kod pro vytvoreni AutoCompleteExtendera
    extenderFirmaCislo = BGCreateComponents.getAutoCompleteExtender("
        GetAutoCompleteDodavateleFaktura", URI_PATH, 3, 500, "ACExCislo", txtCisloFirmy.ID,
        BGNazvy.OBJECT_NAME_FAKTURA_PRIJATA + "Cislo", true);
    ...
    // pridani na panel
    panelTabObecne.Controls.Add(txtCisloFirmy);
    panelTabObecne.Controls.Add(extenderFirmaCislo);
    ...
}
```

Výpis 13: Vytvoření AutoCompleteExtenderu ve třídě BGPopupWindowFaktura

```
// Web Services
[System.Web.Services.WebMethod(EnableSession = true)]
public static string[] GetAutoCompleteDodavateleFaktura(string prefixText, string count, string
    contextKey)
{
    ...
    if (contextKey.Contains("Cislo")) // nastaveni atributu
        atribut_name = " cislo_firmy ";
    ...
    // sestaveni dotazu
    sqlQuery = "SELECT *," + atribut_name + " FROM ViewFirma WHERE (" + atribut_name + "
        like '%@input%')";
    ...
        while (reader.Read())
        {
            Firma myFirma = firmaDB.AutoCompleteFillDataRecord(reader);
            listResults .Add(myFirma.Autocomplete_result + "\t\t\t" + myFirma.Nazev);
        }
    ...
    return listResults .ToArray();
}
```

Výpis 14: Ukázka webové služby AutoCompleteExtenderu z Faktura.aspx.cs



Obrázek 19: Záložka Položky formuláře faktury

Obrázek 19 ukazuje druhou stěžejní záložku detailu faktury pro obsluhu položek. Z viditelných iplementovaných funkcí bych jmenoval přidání položky do gridu. Ruční výběr provede vyvolání záložky zásob, ruční přidání vyhledává zadaný kód a přidání nákladů plní automatickou funkci rozpočítávání částky poměrem mezi položkami. Nad komponentu `BGHtmlGrid` jsou nasazeny JavaScriptové funkce automatických přepočtů sloupečků s grafickým oznámením o změně řádku.

Nad formulářem lze vyvolávat i pomocné vyskakovací okna (zde pro nastavení zaokrouhlení), které jsou založeny na prvku AJAX Control Toolkit `ModalPopupExtender`. V podstatě jde o svázání panelu s extenderem, který poskytne metody pro řízení jeho zobrazení `Show()` a skrytí `Hide()`.

6.2.4 Proces vyvolání nového okna

Proces vyvolání formuláře do nového okna prohlížeče zavádí využití JavaScriptových kódů ukázaných ve výpise 15. Metoda vrací `false`, aby nedošlo k provedení serverové metody, pokud je registrována. Registrace JavaScriptové metody zvolenému tlačítku nad gridem se provádí za pomoci proměnné `Attributes`.

```
...
function openFakturaPrijataInsert() {
    // ZAVEDENI opatreni nemoznosti vyvolani vice oken pod jednim modulem
    if (exist_opened_window(winPolozky)) // pokud okno existuje ukoncit
    {
        alert(hlaskaOknoExistujeNelzeDalsiOtevrit);
        return false;
    }

    // vyvolani noveho okna prohlizece ve stylu PopupOkna
    winPolozky = window.open('PopupWindowFaktura.aspx?agenda=FakturaPrijata&id=-1&mode=
        Insert', Toolbar=no, resizable=No,location=no, directories=no, menubar=no, width=1380,
        height=710');

    return false;
}
...
```

Výpis 15: Ukázka JS kódu vyvolání okna formuláře faktury

6.2.5 Srovnání komponent Infragistics a Telerik

Během práce s ovládacími prvky Infragistics a Telerik se využívaly typově podobné komponenty, především grid a tree. V procesu vývoje došlo k prvotnímu implementování Infragistics prvků. Projevily se tak nedostatky v komunikaci s ostatními ovládacími prvky, a především neúplná a někdy nesprávná funkčnost nabízených metod. Infragistics svoje komponenty na žádosti uživatelů pravidelně aktualizuje, ovšem jde o zásadnější typy chyb.

Telerik je kompatibilní nad většinou známých prohlížečů. Naopak u Infragistics prvků byly problémy u zobrazení i funkčnosti. S obsahem celého balíčku Telerik je k dispozici i reportovací nástroj Telerik Reporting postaven nad SSRS (SQL Server Reporting Services).

Telrik má také propracovanější komponentu grid, která je rychlejší při větším množství záznamů a má rozsáhlejší funkčnost - lépe řešen vlastní výběr sloupců, zobrazení hierarchických dat anebo zobrazení počtu řádku na stránce. U gridu Infragistics byl například problém s uchováváním stavů označených řádků. Navíc komponenta UltraWebTree neposkytovala možnost vázání dat se všemi datovými zdroji, což Telerik umožňuje.

Závěrem lze konstatovat, že Telerik komponenty představují, oproti Infragistics, znatelný rozdíl v jejich užívání, především díky své robustnosti a zároveň spolehlivosti nabízených ovládacích prvků. Mezi další výhody patří rychlost v jejich vykreslování a v práci s načítáním dat do prvků či snadnější přístup k funkcím jak v aplikačním kódu, tak hlavně v kódu JavaScriptovém.

6.3 Vytvoření PDF

Pro realizaci požadované funkce vytvoření PDF souboru nad vybranými záznamem bylo nutné nejdříve najít existující volně dostupný projekt nad .NET frameworkem. Takové kritéria splňovala .NET knihovna `PDFSharp`, která umožňovala tvorbu elementárních prvků PDF dokumentu příkazy pojmenovaných podle jejich účelu vykreslení - např. `Page`, `Line`, `Image`, `Text`.

Proces začíná výběrem záznamu v systému, kde se jeho identifikátor předá obsluze pro vytvoření PDF. Pro získání vybraného byznys objektu byli využiti existující aplikační manažeři (BLL). Sekvenčním zpracováním, řádek po řádku, se provádělo vytváření stránek PDF dokumentu prostřednictvím výše zmíněných příkazů. Výsledný dokument byl vygenerován příkazem `createPdf()` nad instaciovaným objektem `PDFSharpDoc`.

S roustoucími požadavky na složitější strukturu PDF dokumentu a nutnosti řízení jeho vytváření se od knihovny `PDFSharp` upustilo. Řešení opět nabídly Telerik komponenty, jejichž součástí je designový nástroj Telerik Reporting. Telerik reporting nabízí intuitivní designové prostředí založené na panelu nástrojů i s prvky pro vázání dat. Umožňuje, za pomocí připravených funkcí, vytvořit report, jenž poté vygeneruje do formátu PDF (lze i EXCEL, HTML, CVS). K dispozici jsou komponenty typu - textové pole, tabulka, obrázek, panel, graf a další. Tím odpadá nutnost řízení kódu a vytváření šablon, jak tomu bylo u předešlé knihovny `PDFSharp`.

Pro sestrojení PDF dokumentu byl vybrán Telerik reporting nástroj, čímž došlo k urychlení vývoje a ke zkvalitnění vzhledu výsledného souboru. Popis Telerik reporting nástroje nespadá do rozsahu práce, ale samotný nástroj pomohl k vytvoření PDF souborů, jejichž ukázka je zobrazena v příloze dokumentu v sekci C.

7 Testování

Systém se aktuálně nachází v počátku vývojové fáze testování. Procesně došlo ke zpracování všech funkcí vyplývajících ze specifikace požadavků. Realizace prezentační vrstvy navázala na definované knihovny, případně styl jejich vytváření, pro vytvoření záložek a formulářů nad hlavními byznys objekty. Z pohledu implementační fáze byl vývoj systému v dané iteraci ukončen, programátorsky funkčně ověřen a předán fázi testování.

Firma provedla nasazení testovací aplikace nad webovým serverem pro zajištění reálného provozu. Dále obstarala lidské zdroje pro testování funkcionalit z pohledu uživatele bez znalosti kódu. Z manuálního testování vzešly především drobné chyby, jak ze strany procesní, tak vzhledové, přibližně ve stejném poměru.

Jejich opravení nebo přidání doplňujících nových funkcí spadalo pod integrační testování, které ověřilo, že nově přidané funkcionality spolu nekolidují. Následně firma připravuje systémové testování pro pravidelné automatizované testy, především pro simulaci zátěže a stability.

7.1 Návrh optimalizace systému

Z testování funkcionalit nad systémem více uživateli vzešel požadavek na možnou optimalizaci rychlosti odezvy vycházející především z jejich zkušeností práce nad desktopovými aplikacemi. Vzhledem k robustnosti systému v nabízených funkcích a kladení důrazu na jeho vzhled vyplynuly následující možnosti změn při zachování stávajících požadavků:

- **Upřednostnění komponent Telerik**
  Aktuální systém využívá komponenty Telerik a Infragistics, kde srovnání ukázalo vhodnost využití Telerik komponent, jejichž přínosem je právě zrychlení práce. Další optimalizace rychlosti přichází s odstraněním Infragistics knihoven ze systému, čímž dojde i k odstranění jejich řízení v kódech systému BIGGIE.

- **Záložkový návrh**
  Záložky se vyvolávají nad specifickými částmi systému v jedné stránce, kde jich může být zobrazeno několik najednou. To vede ke zpomalení odpovědi serveru na uživatelské akce. Z principu fungování zpětného odeslání (postbacku) se stránka nejdříve celá vytvoří, a až poté dojde ke zpracování asynchronního požadavku. Níže prezentovaný návrh řešení představuje využití plovoucího rámu (iframe) jako standardní html tag.

- **První spuštění**
  Aktuální návrh záložek a formulářů vychází z podpůrných komponent, které se nacházejí ve složce `App_Code`. ASP.NET složky při prvním spuštění dynamicky zkompiluje do jednoho dll souboru, což zabírá více času. Zrychlení představuje

využití předběžné kompilace webu, kterou umožňuje nástroj .NET frameworku aspent_compiler.exe. Předběžná kompilace se doporučuje až po konečném nasazení webové aplikace na server, což vzhledem k aktuálním změnám v kódech systému patří do rysů budoucího zrychlení.

7.1.1 Využití IFRAME

HTML párová značka `iframe` umožňuje vložit do stránky rám o dané velikosti a zobrazit v něm jinou stránku. Využití se nabízí na základě zachování záložkového systému a nutnosti oddělení vyvolávaných požadavků na záložce od ostatních.

Princip návrhu spočívá v tom, že se jednotlivé zpětné odeslání, při obsluze události, provádí na specifické stránce, která je součástí záložky. To vede především ke zrychlení vyřízení požadavku, neboť se při zpětném odeslání neodesílá celá stránka se všemi záložkami a s menu, jako tomu je v aktuálním návrhu systému BIGGIE. Stručný popis řešení:

1. Aktuální obsah záložek separovat do samostatných aspx stránek pro každou agendu (faktury, dodací listy, sklady, karty, atd.).
2. Novým obsahem záložky bude pouze tag `iframe` s odkazem na specifickou stránku přes vlastnost `src`, jak ukazuje výpis 16. To vede k úpravě procesu vyvolání záložky a třídy `BGTab` pro její vytvoření.
3. Nutné přesunutí obsluhy událostí hlavní stránky `Default.aspx.cs` ke specifickým stránkám, následná úprava jejich kódů, i kódů pro obsluhu procesů ve hlavní stránce.

```
...

<cc1:TabPanel runat="server" ID="tab1">
    <ContentTemplate>
   <iframe src="nazevStrankyProAgendu.aspx"></iframe>
    </ContentTemplate>
</cc1:TabPanel>

...
```

Výpis 16: Ukázka šablony pro tag iframe

Práce ve své praktické části poskytla realizované řešení toho problému nad vybranými agendami, což zadávající firmě poskytlo časový odhad možného předělání. Časové výsledky odpovědí serveru na požadavky, zachycenými doplňkem prohlížeče FireBug, odpovídaly separovanému oddělení zpracování požadavků na samotné stránce. Přesné časové odhady by se měli sledovat při nasazení celkového návrhu na server. Obecně lze ovšem ze sledování nad dvěmi otevřenými záložkami konstatovat, že se při požadavku na záložce zrychlila odpověď o více než polovinu.

Aplikovat takové řešení spadá ovšem do dlouhodobého budoucího návrhu, kde se navíc jedná o jednu z několika možností, kterou firma vybere v době realizace optimalizace.

8 Závěr

Diplomová práce se stala součástí produktu ERP Live firmy XEVOS Solutions s.r.o., když utvořila její funkční jádro. Firma se podílela na specifikaci požadavků a určovala použité technologie k vývoji systému. Použitím nejnovějších ASP.NET technologií, s přidanými komponentami, dosáhl celkový dojem ERP systému inovativní a interaktivní koncepce. S využitím knihoven AJAX dostal návrh webového informačního systému praktický význam a webová aplikace se uživatelsky přiblížila aplikaci desktopové.

Jádrem systému se myslí implementace základních agend firmy zabývající se obchodem - jako fakturace, dodací listy, sklady, karty, aj. Interakce mezi agendami a více záložkový návrh zpřehlednili uživatelskou práci. Po funkční stránce obsáhla práce více než základní firemní procesy a poskytla základ pro rozvoj ERP systému.

Práce také poskytla zrealizovaný návrh pro budoucí řešení optimalizace. Především jde o skloubení technologie ASP.NET AJAX s iframe tagem, čím se dosáhlo separování zpracovávaných požadavků nad otevřenými záložkami. Zrychlení se projevilo tím, že vznikla nezávislost mezi jednotlivými zobrazeními a asynchronní vykonávání se týkalo pouze záložky, ve které bylo vyvoláno.

9 Reference

[1] ASLESON, Ryan; SCHUTTA, T. Nathaniel. *AJAX: vytváříme vysoce interaktivní webové aplikace*. 1. vyd. Brno: Computer Press, 2006. ISBN 80-251-1285-3.

[2] TRÁVNÍČEK, Milan. *Správa paměti v jazyce C#*. Ostrava, 2007. Bakalářská práce v rámci Fakulty elektrotechniky a informatiky Vysoké školy báňské - Technické univerzity katedry informatiky. Vedoucí bakalářské práce Petr Šaloun.

[3] LACKO, Luboslav. *Ajax: hotová řešení*. 1. vyd. Brno: Computer Press, 2008. ISBN 978-80-251-2108-5.

[4] *UpdatePanel Control Overview* [online]. c2010, [cit.2010-18-04]. Dostupné z: http://msdn.microsoft.com/en-us/library/bb386454.aspx

[5] FOWLER, Martin. *Destilované UML*. 1. vyd. Praha: Grada, 2009. ISBN 978-80-247-2062-3.

[6] VONDRÁK, Ivo. *Metody specifikace softwarových systému*. Ostrava, 2005.

[7] MACDONALD, Matthew; SZPUSZTA, Mario. *ASP.NET 2.0 a C#: tvorba dynamických stránek profesionálně*. 1. vyd. Brno: Zoner Press, 2006. ISBN 80-86815-38-2.

[8] *Model-View-Controller* [online]. c2010, [cit.2010-18-04]. Dostupné z: http://msdn.microsoft.com/en-us/library/ms978748.aspx

[9] *System.ComponentModel Namespace* [online]. c2010, [cit.2010-18-04]. Dostupné z: http://msdn.microsoft.com/en-us/library/system.componentmodel.aspx

[10] EVJEN, Bill; HANSELMAN, Scott; RADER, Devon.*ASP.NET 3.5 v jazycích C# a Visual Basic : programujeme profesionálně.* . 1. vyd. 1. Brno: Computer Press, 2009. ISBN 978-80-251-2069-9.

A Scénář případů užití

A.1 Stornování objednávky

Scénář případu užití zachycující proces stornování objednávky přijaté pomocí minispecifikace.

Název: UC 3 - Stornování přijaté objednávky

Záměr: Prodejci je zákazníkem předán požadavek na stornování objednaného zboží. Operátor zaeviduje stornování objednávky do systému.

Rozsah: Systém BIGGIE
Úroveň: Uživatelský cíl

Primární aktor: Prodejce

Účastnící a zájmy: Prodejce vyžaduje úspěšné vyhledání stavu objednávky a vložení stornované objednávky do systému

Vstupní podmínka: Prodejce je přihlášen v systému

Minimální záruky: Prodejce je vždy informován o úspěchu či neúspěchu stornovaní objednávky

Záruky úspěchu: V systému je záznam o stornování objednávky

Spouštěč: Prodejce dostal požadavkem na stornování objednávky

Hlavní scénář:

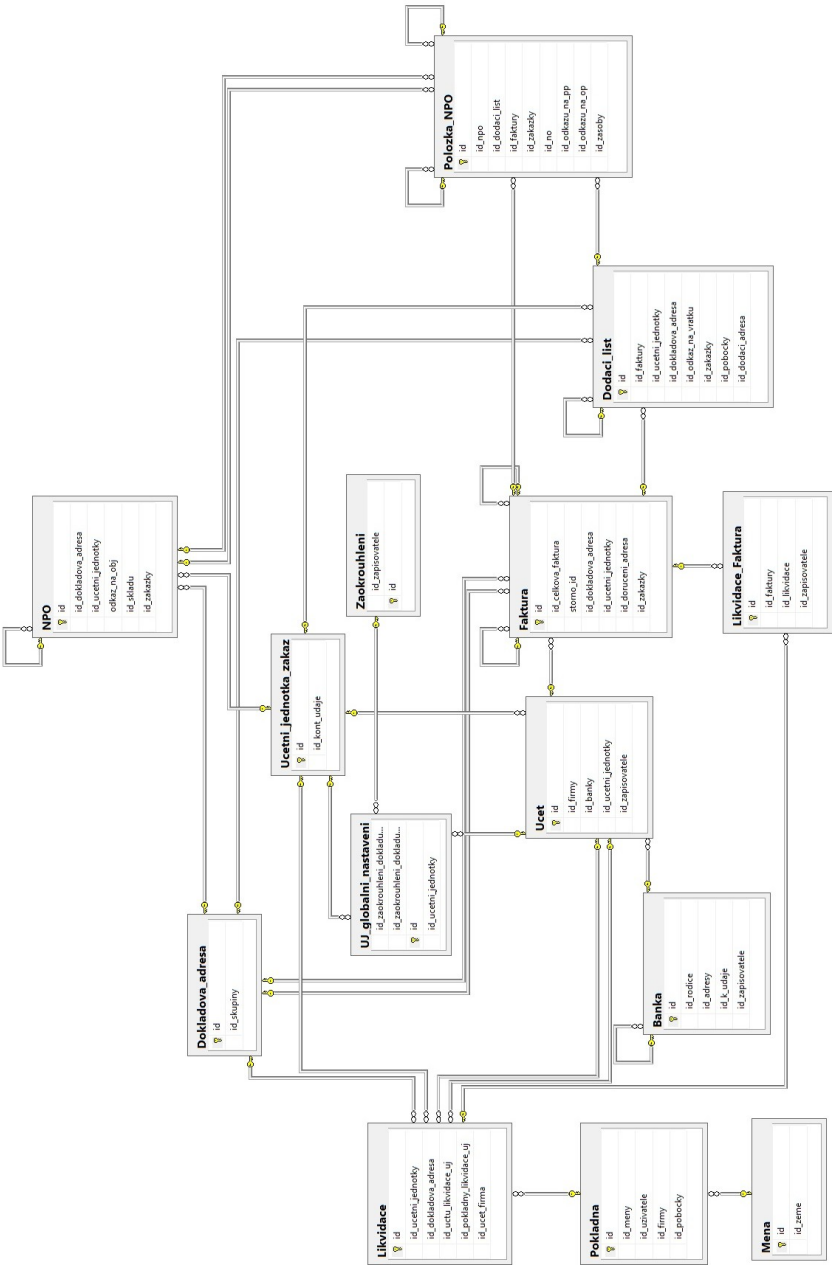
```
1. Prodejce zvolí nabídku pro stornování existující objednávky.
2. Prodejce zadá číslo objednávky do systému.
3. Systém provede zjištění stavu objednávky.
4. Prodejce potvrdí stornování objednávky.
5. Systém vytvoří identickou objednávku vzhledem k nalezené,
přidá opačnou cenu k položkám a záznam uloží.
6. Systém do vyhledané objednávky přidá odkaz na stornovanou
objednávku a záznam uloží.
7.  Systém informuje prodejce o uložení.
8. Prodejce zkontroluje záznam ve výpise objednávek.
```

Rozšíření:

```
1–4a. Prodejce může kdykoliv ukončit vytvoření stornované objednávky.
1–8a. Systém ukončí případ užití bez uložení jakýchkoliv údajů.
2a. Prodejce nezná číslo objednávky.
2a1. Prodejce vyplní název firmy do vyhledávání.
2a2. Prodejce vybere objednávku.
3a. Objednávka má stav doručena.
3a1. Systém předá operátorovi informaci o nemožnosti stornování
 objednávky.
```

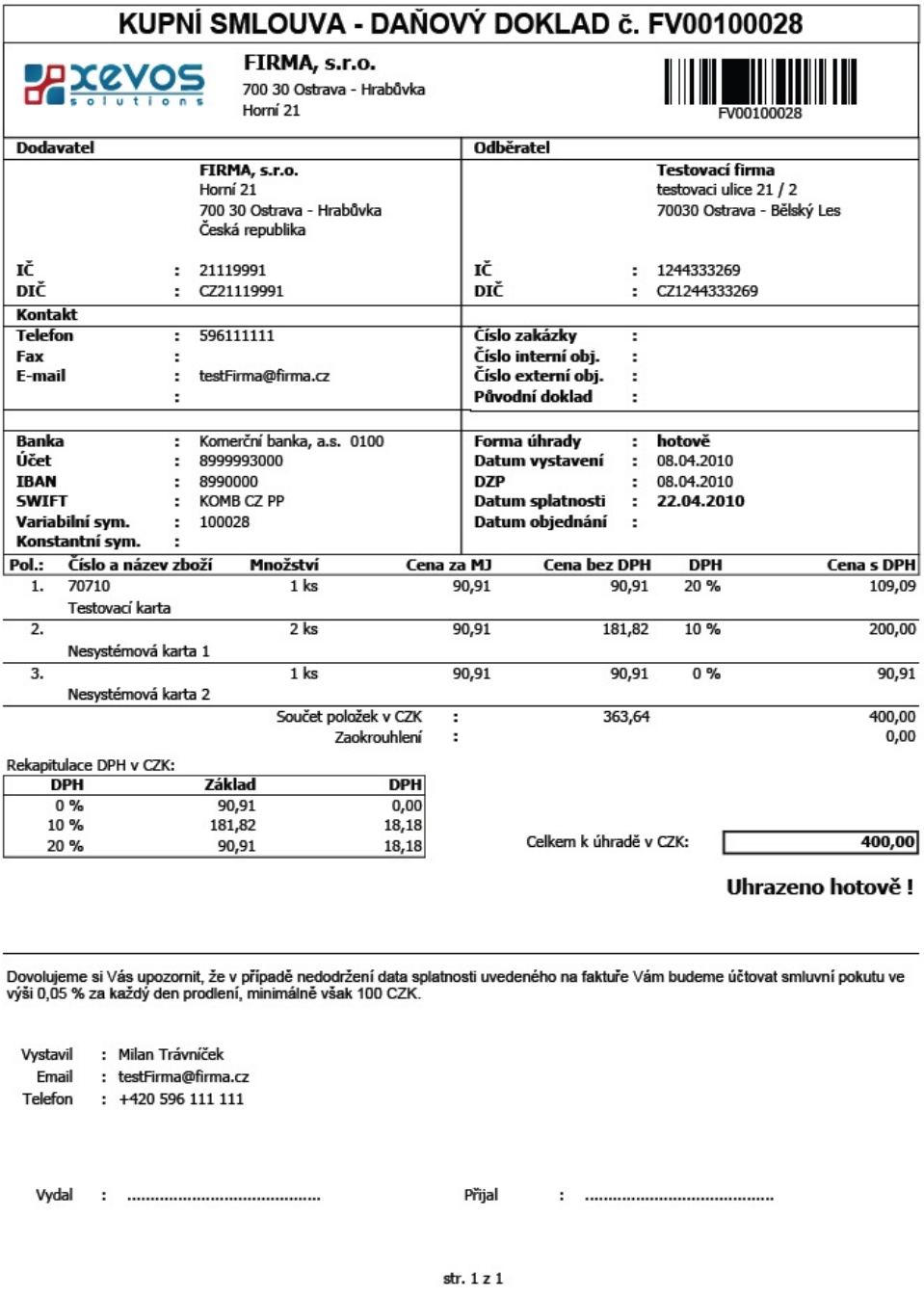
B Návrh

B.1 ER diagram dokladů



Obrázek 20: Návrhový ER diagram dokladů - zobrazení pouze klíčů

C Vytvořené PDF nad systémem



KUPNÍ SMLOUVA - DAŇOVÝ DOKLAD č. FV00100028

xevos solutions

FIRMA, s.r.o.
700 30 Ostrava - Hrabůvka
Horní 21

FV00100028

Dodavatel		Odběratel	
FIRMA, s.r.o. Horní 21 700 30 Ostrava - Hrabůvka Česká republika		**Testovací firma** testovaci ulice 21 / 2 70030 Ostrava - Bělský Les	
IČ : 21119991 **DIČ** : CZ21119991		**IČ** : 1244333269 **DIČ** : CZ1244333269	
Kontakt			
Telefon : 596111111 **Fax** : **E-mail** : testFirma@firma.cz :		**Číslo zakázky** : **Číslo interní obj.** : **Číslo externí obj.** : **Původní doklad** :	
Banka : Komerční banka, a.s. 0100 **Účet** : 8999993000 **IBAN** : 8990000 **SWIFT** : KOMB CZ PP **Variabilní sym.** : 100028 **Konstantní sym.** :		**Forma úhrady** : **hotově** **Datum vystavení** : 08.04.2010 **DZP** : 08.04.2010 **Datum splatnosti** : **22.04.2010** **Datum objednání** :	

Pol.:	Číslo a název zboží	Množství	Cena za MJ	Cena bez DPH	DPH	Cena s DPH
1.	70710 Testovací karta	1 ks	90,91	90,91	20 %	109,09
2.	 Nesystémová karta 1	2 ks	90,91	181,82	10 %	200,00
3.	 Nesystémová karta 2	1 ks	90,91	90,91	0 %	90,91
		Součet položek v CZK	:	363,64		400,00
		Zaokrouhlení	:			0,00

Rekapitulace DPH v CZK:

DPH	Základ	DPH
0 %	90,91	0,00
10 %	181,82	18,18
20 %	90,91	18,18

Celkem k úhradě v CZK: **400,00**

Uhrazeno hotově !

Dovolujeme si Vás upozornit, že v případě nedodržení data splatnosti uvedeného na faktuře Vám budeme účtovat smluvní pokutu ve výši 0,05 % za každý den prodlení, minimálně však 100 CZK.

Vystavil : Milan Trávníček
Email : testFirma@firma.cz
Telefon : +420 596 111 111

Vydal : ..... Přijal : .....

str. 1 z 1

Obrázek 21: PDF detail faktury